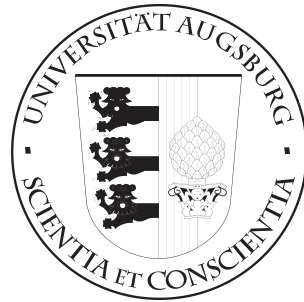


UNIVERSITÄT AUGSBURG



Formal Verification of Information Flow Secure Systems with IFlow

**Peter Fischer, Kuzman Katkalov, Kurt Stenzel,
and Wolfgang Reif**

Report 2012-05

April 2012



INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

Abstract

This report presents an approach called IFlow which allows the model-driven development of secure systems regarding information flow. The approach focuses on the application domain of mobile applications and web services. A developer starts by creating an abstract UML model of a system where he can additionally specify information flow properties the system must satisfy. From the model, Java code is generated together with an information flow policy that can be checked by automated analysis tools like Jif or Joana. In addition, the UML model is transformed into a formal specification which is the basis for formal reasoning within our formal framework including the interactive theorem prover KIV. While automated tools are designed for the simple property of noninterference, formal verification allows to express more complex properties. In order that the results of verification can be carried to the code level and that the results of automated code analysis can be used as lemmas for formal verification, an information flow-preserving refinement relation is established between the formal specification and the code. The focus of this report is on the aspects of formal verification.

1 Introduction

The market for mobile devices, especially smartphones, is rapidly growing. PCs and laptops are getting replaced for common task like browsing the internet, sending emails, or handling business documents. A smartphone offers even more functions, like navigating, talking to other people, taking photos, and listening to music. All these applications have one thing in common: they all require or use data from the phone—data which reveals personal information about the user. This includes confidential information, and aggregated, it constitutes a profile of the user's character and practices.

In order to use even more functions and services on a smartphone, applications which are developed by third-party providers can be downloaded and installed via app stores. Smartphone users become more and more aware of leaving digital fingerprints in the web, and they recognize data privacy as an important issue. With personal information on their phones, apps pose an enormous threat on privacy.

It is very common that gratis apps send personal data to advertisement companies. Recently, several apps were discovered where data is leaked without authorization [1, 4]. Leakage can also occur accidentally due to programming failures. Users wish that, with every downloaded application, they are informed about its privacy policy, and they want guarantees that this policy is satisfied by the application.

With our project called IFlow¹ we contribute to improving the current situation by developing a model-driven software engineering method for privacy-aware systems and integrating information flow analysis. The application domain of IFlow includes mobile applications and services. By modeling security properties already in design phase, a software developer thinks about security from the beginning and on a high level where he is not distracted by details

¹This work is sponsored by the priority programme 1496 “Reliably Secure Software Systems” of the Deutsche Forschungsgemeinschaft DFG.

of the code. The IFlow approach in whole is presented in Section 2. The report focuses on the modeling of systems in Section 3 and on formal verification of information flow properties Section 4. Section 5 concludes the report. [8] addresses the generation of code in detail.

2 The IFlow Approach

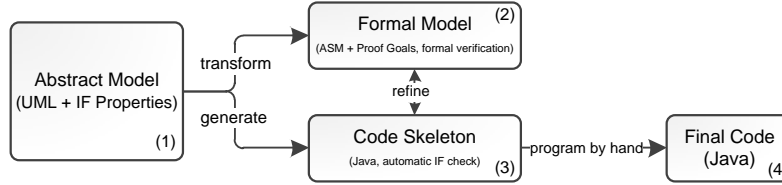


Figure 1: Model driven approach of IFlow.

IFlow is an approach for the model-driven development of privacy-critical, secure systems. It is depicted in Figure 1. In the beginning, a developer models a system (1), which can include mobile applications and web services. When modeling the dynamic part of a system, the developer focuses on the communication between components and on security aspects. The application logic within components does not have to be modeled in full detail. Instead, local functionality can be modeled as a black box. In addition, the developer can specify information flow properties and express confidentiality of data. For modeling, a subset of UML is used, together with a UML profile for approach-specific elements. The UML model is then extended with security annotations depicting an information flow policy.

The platform-independent UML model is the source for two transformations. In one branch, a Java code skeleton is generated (3). The code is analyzed by automated information flow analysis tools. We investigated Jif [11] and Joana [7]. In order to obtain complete Java code, gaps in the code are programmed by hand. Then, the code analysis tool can check the code again for newly introduced information flow and finally be deployed to an Android phone² and to Java EE web services (4).

Since automated code analysis tools can basically express the property of noninterference [5], they are limited when it comes to checking more complex information flow properties. Therefore, in the other transformation branch, a formal model (2) is generated which is integrated with a framework allowing tool-supported verification. We aim to express and prove various properties which include the declassification of certain data and to consider different policies of a program in different phases of lifetime.

The two branches are not treated separately. Instead, a 1:1 information flow-preserving refinement relation is established between the formal specification level and the code level. This enables to carry results of verification to the code level and to lift results of code analysis tools to the formal level where they can be used as lemmas for verification.

²Android is a natural choice as mobile target platform since it is the most widespread mobile operating system and its applications are based on Java.

This report is going to illustrate the formal aspects of the IFlow approach and present the formal framework in more detail.

3 Modeling Information Flow Secure Systems

3.1 Travel Planner Case Study

We explain the IFlow approach with the case study *Travel Planner*, which is a travel booking system consisting of a *travel agency* service (TA) providing flight offers to the user of a mobile *travel planner* (TP) application, developed by the TA. The user is able to select a favored flight offer from a list of offers received from the TA and pay for the flight ticket directly at the *airline* service using the credit card data stored inside a *credit card center* application on his mobile device. The TA then receives a commission from the airline. Secure information flow within this system has to be ensured to provide the user with the guarantee that his credit card data is only ever received by the intended airline, and only after his explicit confirmation.

3.2 Static View

IFlow uses UML to model both static and dynamic views of an application, which can consist of several application agents communicating with each other. We use a class diagram to model smartphone applications, web services or user interfaces, representing each of those agents with a UML class marked with a **Application**-, **Service**- or **User**-stereotype from the predefined IFlow UML profile. Agent class attributes denote data storage of the agent, with their data types also modeled as classes within the class diagram. A **Manual**-stereotyped class contains the signatures for all manual methods (i.e. methods that can later be implemented manually by the developer in order to realize certain application functionality) to be used in sequence diagrams for this application. Figure 2 shows an excerpt from the class diagram for the TravelPlanner case study, picturing all involved agents³.

The IF annotations are modeled as UML constraints on class attributes to define their security level and establish a noninterference property (or several). An annotation consists of a list of IFlow agents being able to observe the annotated attribute. Figure 2 shows the *User* annotation applied to the `creditCardData`-attribute of class `CreditCardCenter`, implying that only the **User** agent is able to read it, while the attribute `flightOffers` of agent **Airline** is annotated with the list *User, TravelAgency, Airline*, making flight offers essentially public to all agents. Such annotations imply a lattice, with information being able to flow only to equally or more restrictively annotated model elements. Agent communication takes place by agents exchanging instances of special message classes, defined by the modeler inside a class diagram by inheriting from an abstract class with the IFlow **Message**-stereotype (as seen in Figure 3 and Figure 9) or predefined in an IFlow UML module (see Figure 11).

³The full model can be found in Figure A and on our website, <http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/projects/iflow/>

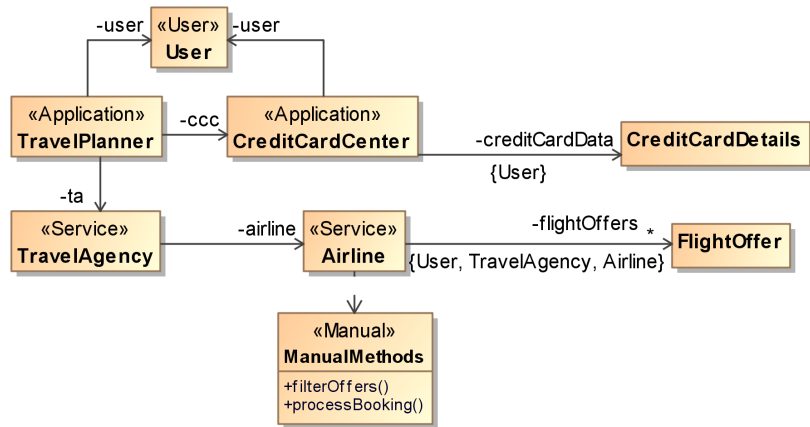


Figure 2: Agent classes

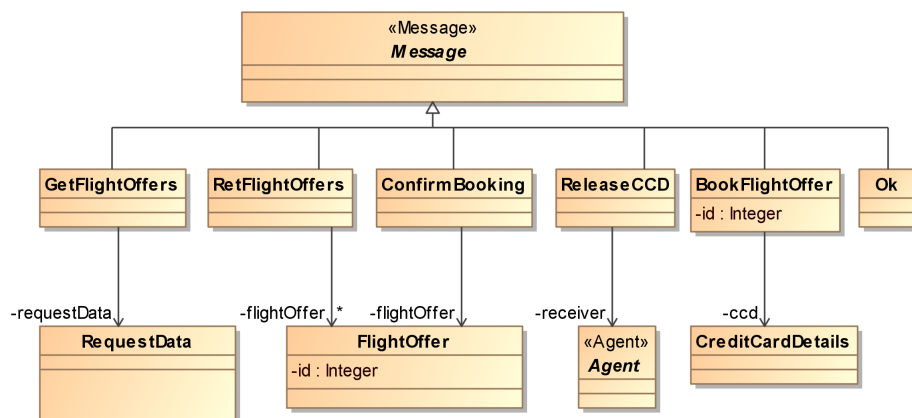


Figure 3: (Excerpt of) message classes

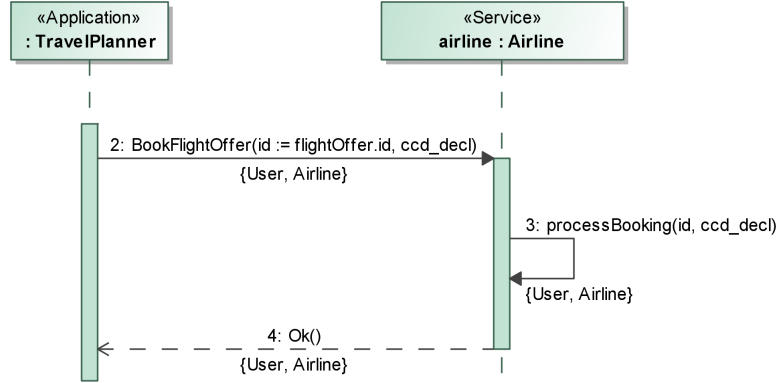


Figure 4: Booking a flight with an airline

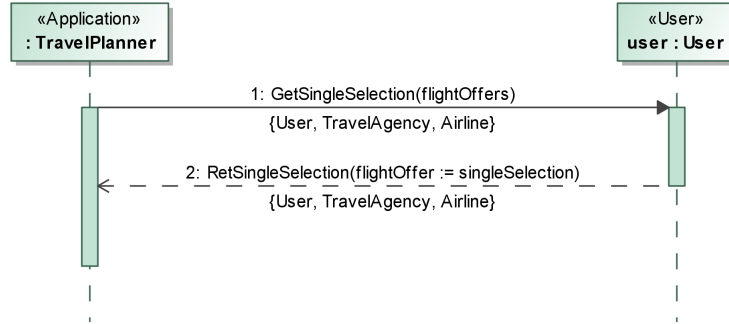


Figure 5: Asking the user to choose an offer from a list

3.3 Dynamic View

Agent communication is modeled with sequence diagrams, with lifelines representing communicating agents. IFlow sequence diagrams utilize a modified version of the Model Extended Language (MEL)⁴ to express message instantiations, method calls or local variable definitions. In order to fix the MEL syntax and the subset of UML usable in IFlow models we defined a metamodel, which is also used to instantiate intermediate IFlow models in the automatic model transformations.

Sequence messages carry the name of appropriate message classes followed by a list of parenthesized lifeline class attributes or local variables, which are used to instantiate the message. The order of those variables is determined by the order of attributes defined in the message class. Since non-primitive attributes of a class are denoted with a named association, and UML class associations are not ordered, we define the following guideline: if a class has exactly one association, it is to be instantiated last. If a class has more than one association, the modeler needs to define a class constructor with an ordered list of parameters carrying

⁴The MEL language was initially designed for the SecureMDD project [10] in order to fully express the functionality of an interactive agent in a UML activity diagram

the same name and type as instantiable class attributes.

The receiving lifeline is then able to access those variables or attributes. Each message must be modeled to receive an answer message, while communication with a user is executed by sending a predefined user message to the user lifeline. Messages received by the user lifeline are meant to represent a dialog- or confirmation messages on the user interface. In order to increase the readability and reusability of certain communication fragments, it is also possible to reference other sequence diagrams by utilizing the UML **Interaction Use** element pointing to the appropriate diagram. Each application task, modeled as the communication between agents must begin with a IFlow-predefined user message from a user lifeline to an application. Manual methods are modeled as UML **Self Messages**; their return value can be assigned to an implicitly declared variable and then be reused as input for other manual methods or as part of a message to another agent.

Figure 4 shows an excerpt from the communication between the TP and the airline, booking a flight by providing the airline with the ID of the flight offer and the user's credit card data. The assignment `id := flightOffer.id` implies that the ID of the offer is being extracted from an attribute of a previously selected flight offer stored in the local variable `flightOffer`. The airline can access the ID via the newly created local variable `id` of type `Integer` (as implicitly derived from the type of the field `flightOffer.id`). It then calls the manual method `processBooking` via a UML **self-message**; the method is defined in the **Manual** class (see Figure 2) in order to process the booking of the flight. It receives both the ID and the credit card data, since no manual method can access local variables or attributes of its callee in order to prevent the developer from introducing new information flows by reading or writing to them.

We annotate each message between IFlow agents in order to specify the security level of data any agent is able to receive. In Figure 4, each message is annotated with `{User, Airline}`, which is more restrictive than the security level of flight offers (see Figure 2) and therefore allows us to send a flight offer ID to the airline. However, the user's credit card data is even more restrictive, which is why it has to be explicitly declassified prior to being sent. This is done in a different sequence diagram after confirming the declassification with the user (see Figure 15).

Figure 5 shows an interaction with the user, with the TP requesting the user to select a flight offer from a previously received list of offers (see Figure 13). To accomplish this, the TP sends the predefined user message `GetSingleSelection` (see Figure 14) containing the list to the user interface. It answers with the user message `RetSingleSelection` containing one element from the list. We model the user selection with a generic attribute of the user class `singleSelection`, which is assigned to an implicitly declared variable `flightOffer` upon receiving of the answer by TP. Its type is derived from the type the elements in the sent list, here `FlightOffer`.

To improve readability and modularity of sequence diagrams, we allow them to reference each other via UML **Interaction Use** elements pointing to the used diagram. We identify the initial diagram used to denote the beginning of an interaction stretching over several sequences by sending a user message from the user lifeline to any IFlow application (**Start** message in Figure 12). To access the sensors of a mobile device we will introduce several predefined

methods, to be used in sequence diagrams as **self messages** just as any manual method. For now, we support the GPS sensor with the predefined methods `getCurrentGPSPos()` and `getAllGPSPos()`; the first returns the current location of the device, while the second provides the application with a list of all locations visited since it has been activated by the user. They have to be annotated in order to identify the initial security level of this data. We support the declassification of data by allowing MEL statements like `y := declassify(x)`, which declassifies data `x` and assigns it the declassification results to `y`. Such statements have to be explicitly annotated with the source and target security levels of the declassification, separated with an arrow (e.g. `User->User,Airline` in Figure 15).

4 Formal Framework

Formal verification provides strong guarantees that a system satisfies certain properties. In this section, we first explain which information flow formalism we chose as a basis for our formal specification and why it conforms well with the properties we want to express. Then, the generic system model, considered by the formalism, is introduced. We created an instance of the system model, which we tailored to our application domain and illustrated with the Travel Planner case study. Analogously, the specification of information flow policies is presented. Experiences of the verification process are reported. The section ends with the argumentation why the refinement relation holds between the formal specification level and the code level.

4.1 Formal System Specification

Our formal framework allows us to formally reason about systems' properties. Therefore, a formal specification of the system and the desired properties are required, which, according to the model-driven paradigm are derived from the UML model.

In the beginning, we want to be able to express noninterference properties by defining an information flow policy for a certain system. After investigating several formalisms and what properties can be expressed by them, we chose to support and use the definition of intransitive noninterference by Ron van der Meyden [14]. Van der Meyden's formalism considered a state-observed model and developed a version with "structured state" based on variables. We also evaluated the noninterference notions of Rushby [12] and MAKs [9]. Rushby only slightly differs from van der Meyden. According to his definition, a state cannot be observed directly but actions produce observable outputs. The MAKs framework considers event-based system models. It allows to express properties about what an observer can learn about the occurrence and non-occurrence of events. Since IFlow considers a setting where the privacy of data is protected, it is natural to choose a formalism where properties of data (rather than events) can be defined. Additionally, van der Meyden allows specifying intransitive policies which generalize the transitive case and can be utilized to define policies that include declassification channels.

According to van der Meyden, a system is defined by a state machine. There is a function `step` where an action causes the transition from one state to

another. He defines a version with “structured state” which refines the abstract notion of state to a set variables. The values of state variables is obtained via the function **contents** which has the signature $state \times name \rightarrow value$, whereas $contents(s, n)$ delivers the value of variable with name **n** in state **s**.

In our approach, the generic system model is refined to match the application domain of several agents communicating with each other. The resulting specification will be illustrated with the instance of the Travel Planner system.

The system’s structure is derived from the UML class diagram which shows agents and their stored data. It is specified using abstract data types [6]. Each type is specified modularly in its own specification extending former specifications and resulting in a graph with dependency edges. UML classes of type **Agent** are mapped to instances of the type **agent** in the formal model. The names of their attributes as well as local variables are captured by the **name**-specification. Since different agents might have variables with same names, the set of names of the original formalism is represented by the cross product $name \times agent$. Variables can be of different types in the UML model (e.g., **CreditCardDetails** and **FlightOffer** in the Travel Planner example from Figure 2). In the formal model these types are subsumed by the type **data** which is a data format where any information relevant for information flow security is defined recursively. The **data** specification for the case study is depicted in Figure 6. Instead of the static function **contents**, we store the system state in a higher order variable mapping $name \times agent \rightarrow data$. Assumed that the system state is represented by variable **content**, $content(n, a)$ returns the value of the variable with name **n** of agent **a** in the current state.

The dynamic part of the system is modeled with UML sequence diagrams. The UML messages are named according to message data types which were defined earlier in a class diagram (see Figure 3). These messages serve as a container for the transmitted data. Hence, they are also included in the **data** specification in Figure 6. On the one hand, messages are containers for data transmission, on the other hand, they induce an action to the receiving agent. The names for the corresponding actions are derived from the message names but have to be unique; or more concrete: one message can be used to send data both from agent A to B and from agent C to D, whereas distinguishable actions have to be inferred. The actions of the system are defined in specification **command**⁵.

In the formal model, the behavior of the system is expressed with an abstract state machine (ASM) [3] which is defined in the corresponding specification **ASM**. The specification of the system’s behavior builds on its structural specification. For each possible action in the system, one ASM rule is defined. Whenever a certain action occurs, the corresponding ASM rule is executed. The scope of an actions reaches from the reception of a message to the sending of an answer. If, in a sequence diagram, there is a manual method between the incoming and the return message, two actions (and two ASM rules) originate: the first action receives the message and the second calls the manual method and returns an answer. Manual methods are mapped to functions with signatures which are equivalent to those of the original methods. They are declared in the **localMethods** specification.

When a message is transmitted, it is stored into an inbox of the receiving

⁵The terms “action” and “command” are used synonymously in this report.

```

data =
data specification
  using command, string-less, nat-mult, agent
  data = CreditCardDetails (. .data : string ;) with isCreditCardDetails
    | FlightOffer (. .id : data ; . .airline : agent ;) with isFlightOffer
    | RequestData (. .data : string ;) with isRequestData
    | GetFlightOffers (. .requestData : data ;) with isGetFlightOffers
    | RetFlightOffers (. .flightOffers : data ;) with isRetFlightOffers
    | ReleaseCCD (. .receiver : data ;) with isReleaseCCD
    | DeclassifiedCCD (. .ccd : data ;) with isDeclassifiedCCD
    | BookFlightOffer (. .id : data ; . .ccd : data ;) with isBookFlightOffer
    | ConfirmBooking (. .id : data ;)
    | PayCommission (. .id : data ;)
    | DataList (. .list : dataList ;) with isDataList
    | Input (. .at : nat → data ;) with isInput
    | InputCounter (. .nr : nat ;) with isInputCounter
    | Boolean (. .boolean : bool ;) with isBoolean
    | Number (. .number : nat ;) with isNumber
    | Command (. .cmd : command ;) with isCommand
    | Idle
    | Agent (. .agent : agent ;) with isAgent
    | null
    | Start
    | GetInputRequestData
    | RetInputRequestData (. .input : data ;) with isRetInputRequestData
    | GetSingleSelection (. .selection : data ;) with isGetSingleSelection
    | RetSingleSelection (. .selection : data ;) with isRetSingleSelection
    | GetMultipleSelection (. .selection : data ;) with isGetMultipleSelection
    | RetMultipleSelection (. .selection : data ;) with isRetMultipleSelection
    | ConfirmRelease (. .what : data ; . .receiver : data ;) with isConfirmRelease
    | RetConfirmation
    | Show (. .data : data ;) with isShow
    | Ok
  ;
  dataList = []
    | . + . prio 9 (. .first : data ; . .rest : dataList ;) prio 9 ;
end data specification

```

Figure 6: Data specification

agent. Therefore, inboxes appear in the **data** specification. An agent has an inbox for each action it can handle. However, an action is only processed if the action is enabled in the agent's state and there is data written to the corresponding inbox. Every agent has got its own internal state which is defined implicitly in the sequence diagrams. The possible states of an agent are computed automatically from the UML model. Initially, an agent is in state "idle" and is waiting for an action which is enabled in this state. When the agent processes an action, its state changes into another state. The new state either expects another action to be processed locally, or, if it sends a message to another agent, it waits for an answer. When the agent sends a return message of the initial message, it finally sets its state to "idle" again. The transitions of an agents internal state are defined by one ASM rule for each agent. The relation between actions and the internal state, in which they are accepted, is coded in the **acceptedCmds** specification.

To illustrate the specification, an excerpt of ASM rules is shown in Listing 1. The first two ASM rules correspond to actions which originate from Figure 4. In ASM rule **BOOKFLIGHTOFFER**, in the beginning, function **Airline_waiting4** checks if the airline accepts the action **BookFlightOfferCmd** in its current state and if the inbox for the action is non-empty. If so, the action is executed and the arguments of the transmitted message are written from the inbox into local variables of the airline. By calling **AIRLINE_NEXT**, the internal state is set. Since a local method is modeled between the original and its return message, there is no new message sent in the end of this action. For ASM rule **AIRLINE_NEXT** there is no corresponding action. Instead, it is responsible for the state transitions of the airline. Dependent of the action which was just processed a new state is set. Since the action was executed, the message is also deleted from the inbox. By calling **AIRLINE_NEXT** in the context of **BOOKFLIGHTOFFER**, the inbox for **BookFlightOfferCmd** is emptied and the state is set to accepting only the action **ProcessBookingCmd** (see lines 11 and 12). When the latter action occurs, corresponding ASM rule **PROCESSBOOKING** is called (lines 22 ff.). Function **Airline_waiting4** again checks whether the action is accepted in the current state of airline. For this action, there is no additional check if something is written into an inbox since the action does not directly follow the reception of a message. There is no inbox to read from. In line 24, local method **processBooking** is called according to the method call from Figure 4. The subsequent status transition sets the status to **Idle** (see line 15 in **AIRLINE_NEXT**). No inbox must be emptied. After the method call a return message is sent back to the travel planner which happens in line 25. A message called **OK** (which contains no arguments) is returned. Since this message is typically used as an empty return message, the action's name differs from the message name to be unique.

Listing 1: ASM rules which are derived from sequence diagrams

1	BOOKFLIGHTOFFER {
3	if (Airline_waiting4 (BookFlightOfferCmd , content)) then {
5	content (id , airline) := (content (inbox.BookFlightOfferCmd ,
7	airline)). id ;
	content (ccd_decl , airline) := (content (inbox.BookFlightOfferCmd
	, airline)). ccd_decl ;
	AIRLINE_NEXT ;
	}
	}

```

9  AIRLINE_NEXT {
11    if(action = BookFlightOfferCmd) then {
13      content(inbox_BookFlightOfferCmd, airline) := null;
15      content(waiting4, airline) := Command(ProcessBookingCmd);
17    }
19    else if(cmd = ProcessBookingCmd) then {
21      content(waiting4, airline) := Idle;
23    }
25    else {
27      ...
    }
  }
}

PROCESSBOOKING {
  if(Airline_waiting4(ProcessBookingCmd, content)) then {
    processBooking(content(flightOffer, airline), content(ccd_decl,
    airline));
    AIRLINE_NEXT;
    content(inbox_OkBookFlightOfferCmd, travelPlanner) := Ok;
  }
}

```

In our approach, we also model user interaction (e.g., in Figure 5). This is modeled with a message sent from an application to the user and a return message. From these message, two ASM rules originate. Actual user input is simulated by providing input arrays of arbitrary length for each action. To capture information flow in a realistic sense, the returned value of the user shall be dependent of the displayed data. This means for Figure 5 that the chosen flightOffer depends on the displayed list of flightOffers as well as on the user's choice. In specification **userMethods**, corresponding functions are defined to capture user interaction. In ASM rule **GETSINGLESELECTION** (see Figure 2), first, a new user input is fetched from the array of inputs for the action **GetSingleSelectionCmd** by calling **PROVIDEINPUT.GETSINGLESELECTION**. Function **selectOne** is declared in **userMethods**. It is called with the received list of elements and the current input, constituting the choice, as arguments returning the chosen element. Of course, the user has no status. However, with the call **USER.FINISHED** the inbox is emptied which, in the figurative sense, means that the user dialog is closed.

Listing 2: ASM rules which are derived from sequence diagrams

```

GETSINGLESELECTION : GETSINGLESELECTION {
2  if (User_received(GetSingleSelectionCmd, content)) then {
4    PROVIDEINPUT.GETSINGLESELECTION;
    content(inbox_RetSingleSelectionCmd, travelPlanner) :=
      RetSingleSelection(selectOne((
        contentinbox_GetSingleSelectionCmd, user).selection,
        content(input(GetSingleSelectionCmd), user)));
6    USER.FINISHED;
8  }
}

```

The full system specification of the travel planner application is shown in Section B.2.

4.2 Specification of Information Flow Policies

Noninterference is a property to express information flow security and was introduced by Goguen/Meseguer [5]. For the canonical case with two security domains, one with high confidentiality (H) and one with low confidentiality (L), it says that actions happening in the high domain can neither be observed directly nor anything about them can be learned at all. When extending the noninterference property to data, the requirement is that variables written in the high domain must not be read in the low domain. The canonical policy $L \rightsquigarrow H$, $H \not\rightsquigarrow L$ is sufficient for theoretical considerations. For real-world applications, a policy generally requires more domains. In our approach, domains are defined as sets of agents which are able to observe data of this particular domain.

After the domains have been defined, an information flow policy is constituted by the following specifications:

- The interference relation \rightsquigarrow (with signature: $domain \times domain$) expresses from which domain information may flow to which domain. It can be derived from the security lattice of the UML model and the set of agents forming a domain.
- **observe** and **alter**, which we defined as relations (with signatures: $domain \times name \times agent$), express which variables can be observed/alterd by a domain. The specification is obtained from the security annotations of attributes and inferred for local variables, depending on the security level of the right hand side of the first assignment.
- The observation function **obs** (original signature: $state \times domain \rightarrow observation$) defines what observation one domain can make in a state. We defined the observation as a subset of the system state. The observable subset for a domain is obtained using the **observe** relation.
- The function **dom** (with signature $action \rightarrow domain$) describes the visibility of an action and also what variables can be read inside it. The specification is obtained from the security annotation of the messages in the sequence diagrams.

Defining a noninterference policy is a challenging task. Finding a suitable set of domains is the most important prerequisite to specify a property of noninterference. Different properties require different sets of domains. It is not clear how to derive a policy systematically.

For us, the following strategy was successful:

1. Inspect variables in the application and annotate which agents may observe it.
2. Examine the actions and choose a domain (defined by a set of agents). Since an action reads and writes variables, the action's domain must allow this and be consistent with the variables' annotations from step 1. This defines the set of domains and the mapping from actions to domains for function **dom**. The set of agents of each domain implicitly defines the interference relation \rightsquigarrow .

3. The **observe** and **alter** relations are then calculated. This requires us to examine each action. Whenever a variable is read or written, it is added to the **observe**- or **alter**- relation of the action's domain respectively. This results in a minimum required set of observable/alterable variables for each domain. Since the observe-/alter-relations are calculated from the application, they must be checked for adequacy.

A transitive noninterference policy in many cases is too restrictive and must be weakened by allowing the declassification of data into a lower. For example, the travel planner declassifies **creditCardData** to enable the booking of a flight. The MEL syntax in our approach allows to introduce explicit declassify-statements in sequence diagrams. The statement is treated as an additional action. The action is mapped to a new domain, which serves as a declassification channel. At this point, the intransitivity of van der Meyden's formalism is exploited. The following scenario illustrates the policy changes: Let **A** and **B** be two domains with $A \not\sim B$. Variable **x** is alterable by **A**. If the content of **x** is read in domain **B**, the policy becomes inconsistent. The solution is to use the declassify statement $x_decl := declassify(x)$ which is executed in an action with associated domain **D**. The domain interference specification is extended by $A \sim D$ and $D \sim B$. Domain **D** may observe **x** and alter **x_decl** whereas domain **B** may observe **x_decl**.

For the travel planner example, we started by requiring that **creditCardData** is not sent to the travel agency. There is no differentiation in this requirement about information flow between the (smartphone) applications **TravelPlanner** and **CreditCardCenter** and what is displayed to the user. Therefore, no differentiation was made; if a domain contains the agent **User** this means that the domain is also visible for the applications on the smartphone.

The following domains originated (the corresponding domain names in the formal model are written in parenthesis):

- All actions that do not concern **creditCardData** are assigned domain **{User, TravelAgency, Airline}** (*User-TA-A-Dom*).
- The actions (and variables) related to **creditCardData** are assigned domain **{User}** (*User-Decl2A-Dom*).
- Before the booking process, **creditCardData** must be declassified. Therefore, a declassification domain is introduced allowing flow from **{User}** to **{User, Airline}** (*Decl2A-Dom*).
- When **creditCardData** is sent to the airline during the booking process, the corresponding domain is **{User, Airline}** (*User-A-Dom*).

Specification **domInterference**, depicted in Figure B.3, shows the implied interferences between domains.

The complete specification of the policy can be found in in Section B.3.

The specification of the formal system model has been derived systematically from the UML model by hand. The goal of our model-driven approach is to automate the transformation. The transformation is split in two steps: First, the UML model is parsed into the intermediate MEL model using Java. The UML model is interpreted for the IFlow approach and is already extended

```

domInterference =
enrich domain with
    predicates .  $\Rightarrow$  . : domain  $\times$  domain;

axioms

    reflexivity : domvar  $\Rightarrow$  domvar;
    // transitive policy:
    User_TA_A_Dom  $\Rightarrow$  User_A_Dom;
    User_TA_A_Dom  $\Rightarrow$  User_Decl2A_Dom;
    User_A_Dom  $\Rightarrow$  User_Decl2A_Dom;
    // intransitive policy with declassification domain:
    User_Decl2A_Dom  $\Rightarrow$  Decl2A_Dom;
    Decl2A_Dom  $\Rightarrow$  User_A_Dom;

end enrich

```

Figure 7: Specification of domain interferences

with MEL statements. The MEL meta model defines the semantics of the modeling approach, according to the descriptions of Figure 3. Xpand is used⁶ to transform the MEL model with a model-to-text transformation into the textual representation of the final system specification.

In KIV, a system is specified in a modular way. This allows to be flexible regarding the exchange of certain parts, e.g., the policy. For the same reason, another formalism can be implemented in the specification. Once the transformations have been established, the model-driven approach allows to support multiple formalisms. Different properties can be defined with different formalisms and the specification is enriched with the desired ones individually for each case. We consider to integrate the MAKES framework [9] as it is very powerful to express properties about deductions of occurrence and non-occurrence of events.

4.3 Proving Information Flow Properties

For the noninterference property, security is defined using the so-called purge-function. A program execution is represented by a sequence of actions. The purge-function deletes, from a sequence of actions, those actions which influence the observation of a particular domain. In the transitive case, all actions are deleted which domain directly interferes the observing domain. In the intransitive variant of the purge-function, all actions are deleted which can directly or indirectly interfere with the observing domain within the given action sequence. If for every domain holds that the resulting purged subsequence is indistinguishable from the original sequence, then the system is considered secure. Van der Meyden presents so-called unwinding conditions. If these conditions hold after each executed action, the system is secure. With the unwinding conditions each

⁶Xpand, <http://www.eclipse.org/modeling/m2t/?project=xpand>

action can be considered locally, instead of inspecting program traces. This reduces the verification effort enormously. For van der Meyden’s variant with a “structured state”, he gives the so-called reference monitor assumptions (RMA) and the alter-observe-interference condition (AOI), which imply the unwinding conditions and thereby security as well.

At first, we specified the generic noninterference formalism as given by van der Meyden [14] with the interactive theorem prover KIV [2]. Subsequently, we proved that the RMAs and AOI imply the unwinding conditions and that the unwinding conditions imply the security notion based on the purge function. We derived the application model for the travel planner case study and specified the policy as described above. Then, we proved that the travel planner model is an instantiation of the generic system model of van der Meyden.

Finally, we proved that the RMAs and AOI hold for the specified system and the policy. The definitions of the conditions look in our system as follows:

- RMA1:

$$\text{eqd}(c, \text{domvar}, c_1) \vdash \text{obs}(c, \text{domvar}) = \text{obs}(c_1, \text{domvar})$$
- RMA2:

$$\vdash \text{eqd}(c, \text{dom}(\text{co}), c_0) \wedge c(\text{nam}, \text{ag}) = c_0(\text{nam}, \text{ag}) \wedge \text{alter}(\text{dom}(\text{co}), \text{nam}, \text{ag})$$

$$\rightarrow \langle \text{STEP\#} \rangle \langle \text{STEP\#} \rangle c_2(\text{nam}, \text{ag}) = c_3(\text{nam}, \text{ag})$$
- RMA3:

$$\vdash \neg \langle \text{STEP\#} \rangle c_2(\text{nam}, \text{ag}) = c(\text{nam}, \text{ag}) \rightarrow \text{alter}(\text{dom}(\text{co}), \text{nam}, \text{ag})$$

RMA 1 says that if the observable variables of two states appear equal to an observing domain then the observations that can be made in both states are the same. Since we defined the observation to be the observable part of the state, this condition trivially holds. RMA 2 and RMA 3 basically ensure that the sets of the observable and alterable variables respectively are chosen big enough w.r.t. the actions reading and writing from variables. For these proofs, the condition has to be satisfied for each action. Using heuristics, the proofs were performed fully automated by KIV. AOI then ensures that the sets of observable and alterable variables do not violate the domain interference policy, i.e., if a variable is alterable in domain d_1 and observable in domain d_2 then there must be interference $d_1 \rightsquigarrow d_2$. For this proof all combinations of domain interferences have to be considered. Hence, the proof effort grows quadratically with the number of security domains.

During verification, we learned some issues concerning the system and the policy specification: The scope of actions was not suitable for all cases. If an action writes into the inbox of the following, more confidential action, the policy is violated. Instead, a new action for sending a message has to be defined, so that, writing to the inbox can happen with the same confidentiality as reading from it. In the travel planner application, we introduced sending actions where they were needed, but we consider introducing generic sending actions. We also discovered the problem that the internal state of each agent is read and written by every action processed by the agent. Hence, all domains correlating to the actions interfere each other. Actually, the states just ensure the defined order of actions of the application which is modeled with the sequence diagrams. The states are assigned only constant values, which do not depend on the variables

of the system, and, thus, they do not encode conditional branching. As a first solution, we extended the required domain interferences and proved that the system satisfies the weaker policy. In future, we will exclude these internal states from the system state. If this is not possible for any reason, we will constrain what may flow through these newly introduced domain interferences. This is a property which can be proven using partial equivalence relations and bisimulation [13].

5 Conclusion

In this report, a brief overview of the model-driven approach, called IFlow, was given and the modeling guidelines were explained. The formal aspects of the approach and the establishment of a formal framework were then presented in more detail. This includes the definition of a model for formal specifications and their systematic derivation from an IFlow UML model. Further, the specification and verification of information flow properties within our framework were presented. For this process, we provided guidelines resulting from personal experiences.

We found that defining an appropriate information flow policy for the non-interference property is quite difficult. On the one hand, all requirements which are imposed on the system must be captured; on the other hand, they have to be precise in order to not reject intuitively secure systems. We learned that a policy, in contrast to our expectations, often is not satisfied by a system since subtle information flows are easily overlooked. While specification of information flow policies was challenging, verification was surprisingly straight forward. Since our verification tool reached a high degree of automation for the proofs, the required effort was very low. If a policy was not satisfiable by the system, verification provided helpful feedback by pointing to problematic spots of the information flow specification. Thus, an iterative process was established where verification provided quick feedback and the policy could be refined continuously.

In the future, we will program an automatic transformation which generates the formal specification from a UML model. We want to be able to prove more properties with information flow policies and declassification channels which are restricted in the *what*- and *where*-dimensions [13]. According to our overall approach Figure 1, a refinement relation between the formal specification and the code level will be established formally and property equivalence between code analysis tools (see [8]) and our model-based formalism are investigated in order to integrate verification with automated checkers on code level.

6 Bibliography

- [1] M. Balanza, K. Alintanahin, O. Abendan, J. Dizon, and B. Caraig. Droid-dreamlight lurks behind legitimate android apps. In *Malicious and Unwanted Software (MALWARE)*, 2011 6th International Conference on, pages 73 –78, oct. 2011.
- [2] Michael Balser, Wolfgang Reif, Gerhard Schellhorn, and Kurt Stenzel. KIV 3.0 for Provably Correct Systems. In Dieter Hutter, Werner Stephan, Paolo

- Traverso, and Markus Ullmann, editors, *Proc. Int. Wsh. Applied Formal Methods*, volume 1641 of *LNCS*, pages 330–337. Springer, 1999.
- [3] E. Börger. The abstract state machines method for high-level system design and analysis. *Formal Methods: State of the Art and New Directions*, pages 79–116, 2010.
 - [4] William Enck, Damien Ocateau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *Proceedings of the 20th USENIX conference on Security*, SEC’11, pages 21–21, Berkeley, CA, USA, 2011. USENIX Association.
 - [5] J.A. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and privacy*, volume 12, 1982.
 - [6] Y. Gurevich. Evolving algebras 1993: Lipari guide. *Specification and validation methods*, pages 9–36, 1995.
 - [7] Christian Hammer and Gregor Snelting. Flow-sensitive, context-sensitive, and object-sensitive information flow control based on program dependence graphs. *International Journal of Information Security*, 8(6):399–422, December 2009. Supersedes ISSSE and ISoLA 2006.
 - [8] K. Katkalov, P. Fischer, K. Stenzel, and W. Reif. Model-Driven Code Generation of Information Flow Secure Systems with IFlow. Technical Report 2012-04, Universität Augsburg, 2012. URL: <http://www.informatik.uni-augsburg.de/lehrstuehle/swt/se/publications/>.
 - [9] H. Mantel. *A uniform framework for the formal specification and verification of information flow security*. PhD thesis, Universitätsbibliothek, 2003.
 - [10] N. Moebius, K. Stenzel, H. Grandy, and W. Reif. ‘SecureMDD: A model-driven development method for secure smart card applications’. In *Availability, Reliability and Security, 2009. ARES ’09. International Conference on*, pages 841–846, march 2009.
 - [11] A.C. Myers, L. Zheng, S. Zdancewic, S. Chong, and N. Nystrom. Jif: Java information flow. *Software release. Located at <http://www.cs.cornell.edu/jif>*, 2005, 2001.
 - [12] J. Rushby. *Noninterference, transitivity, and channel-control security policies*. Citeseer, 1992.
 - [13] A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *CSFW*, pages 255–269. IEEE Computer Society, 2005.
 - [14] R. van der Meyden. What, indeed, is intransitive noninterference? *Computer Security-ESORICS 2007*, pages 235–250, 2007.

A Travel Planner UML Model

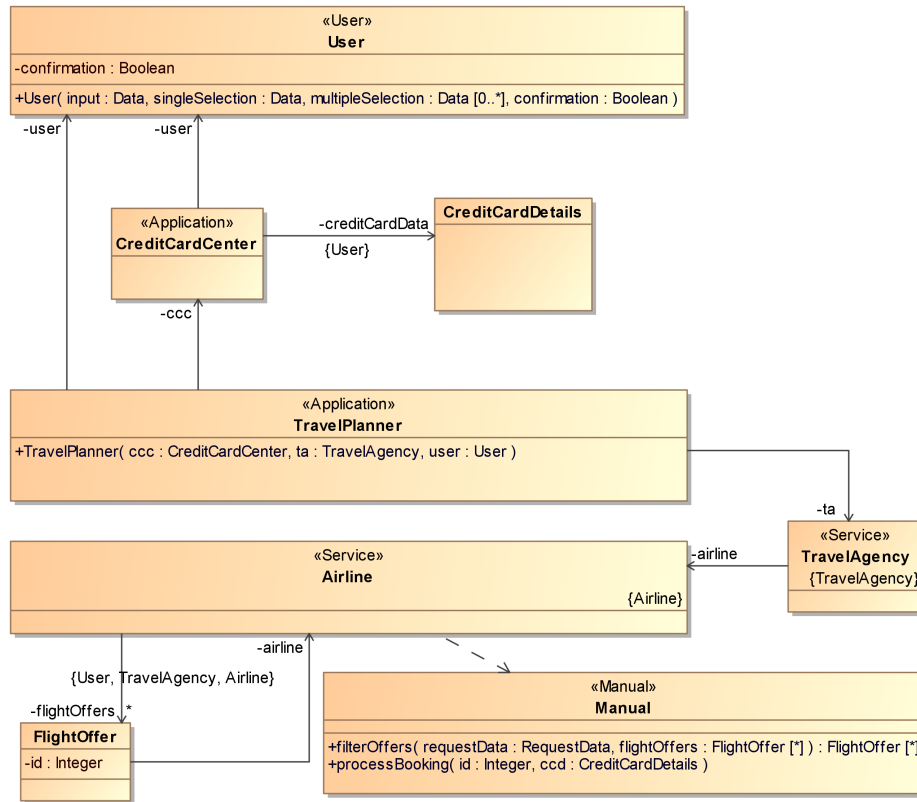


Figure 8: Application component diagram

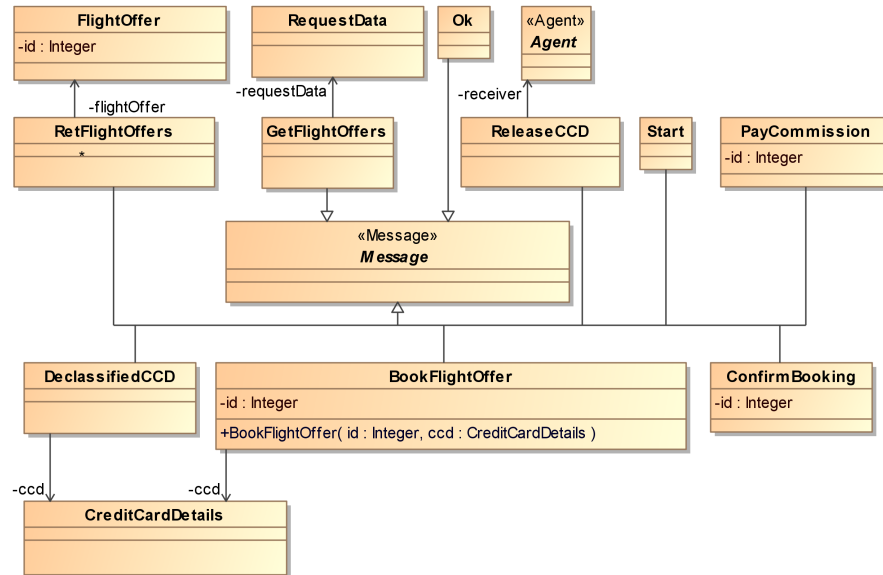


Figure 9: Application messages diagram

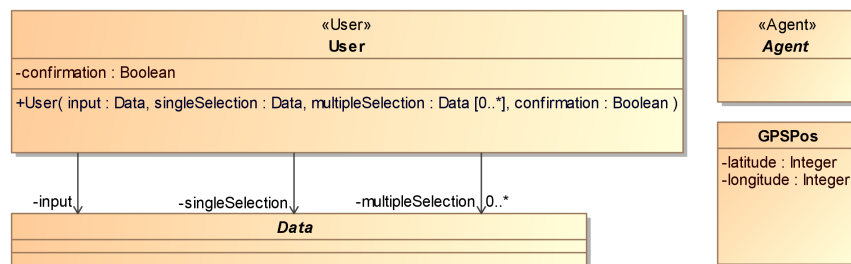


Figure 10: Standard classes

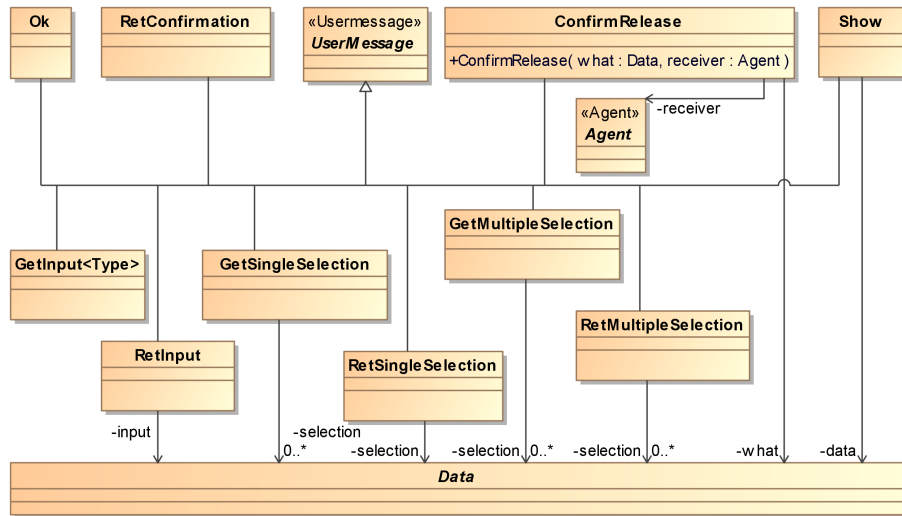


Figure 11: Standard user messages

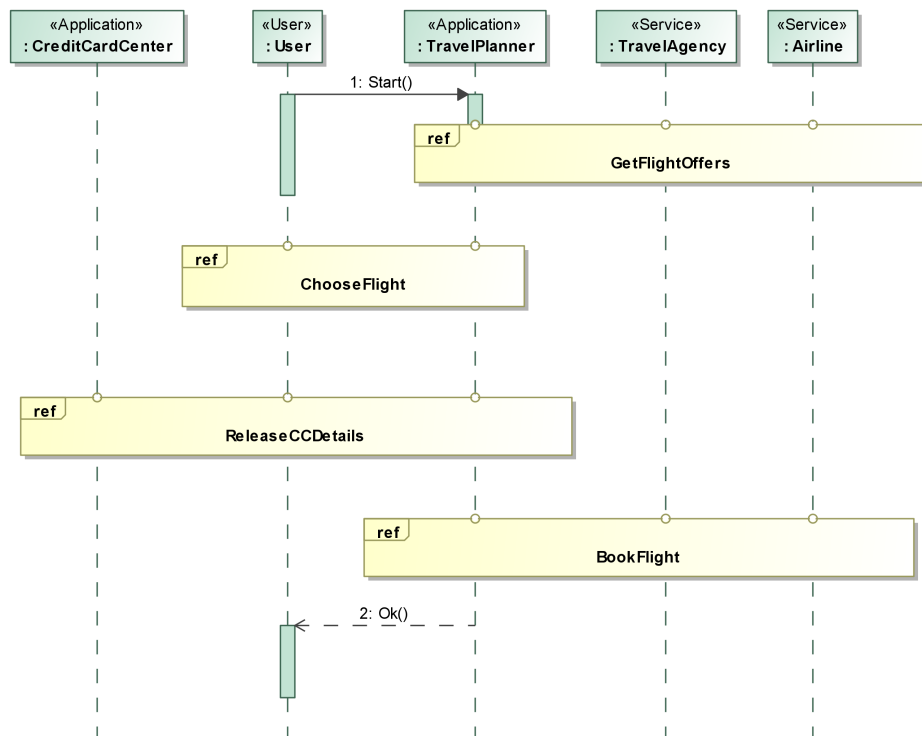


Figure 12: Behavior overview diagram

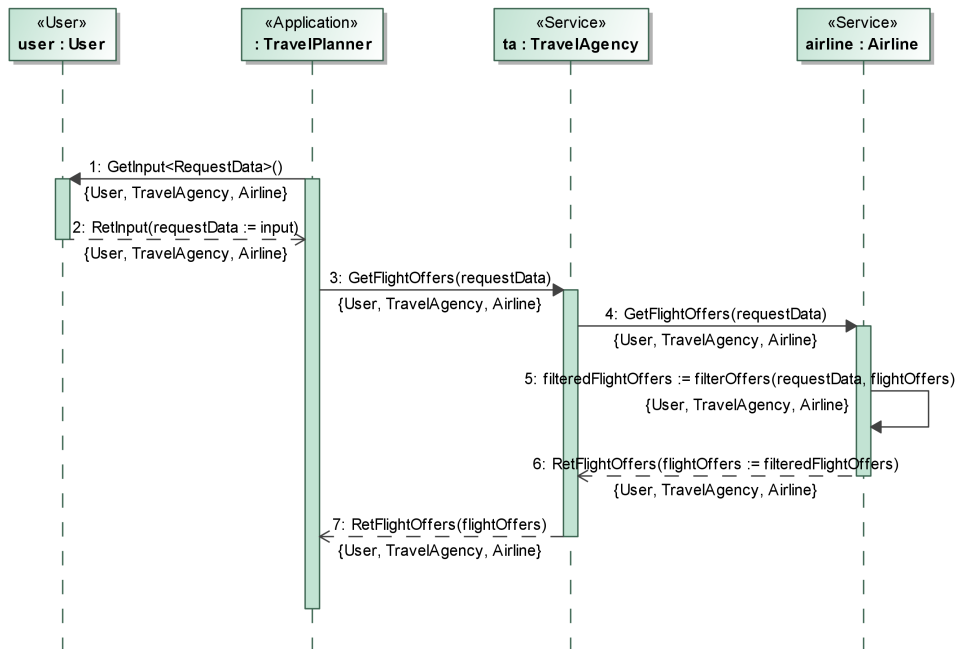


Figure 13: Request flight offers

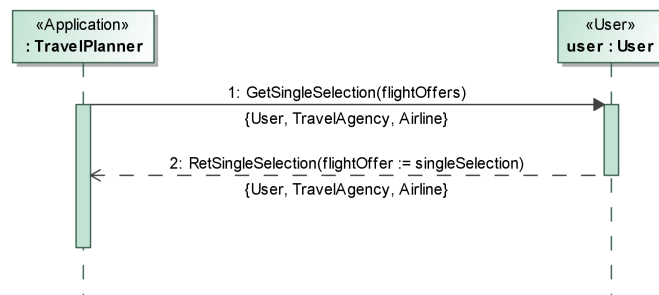


Figure 14: Choose a flight

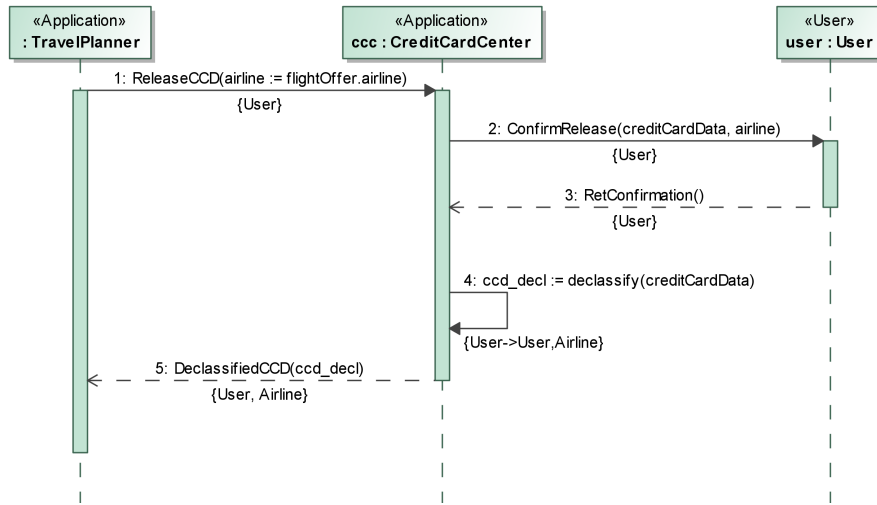


Figure 15: Release credit card details

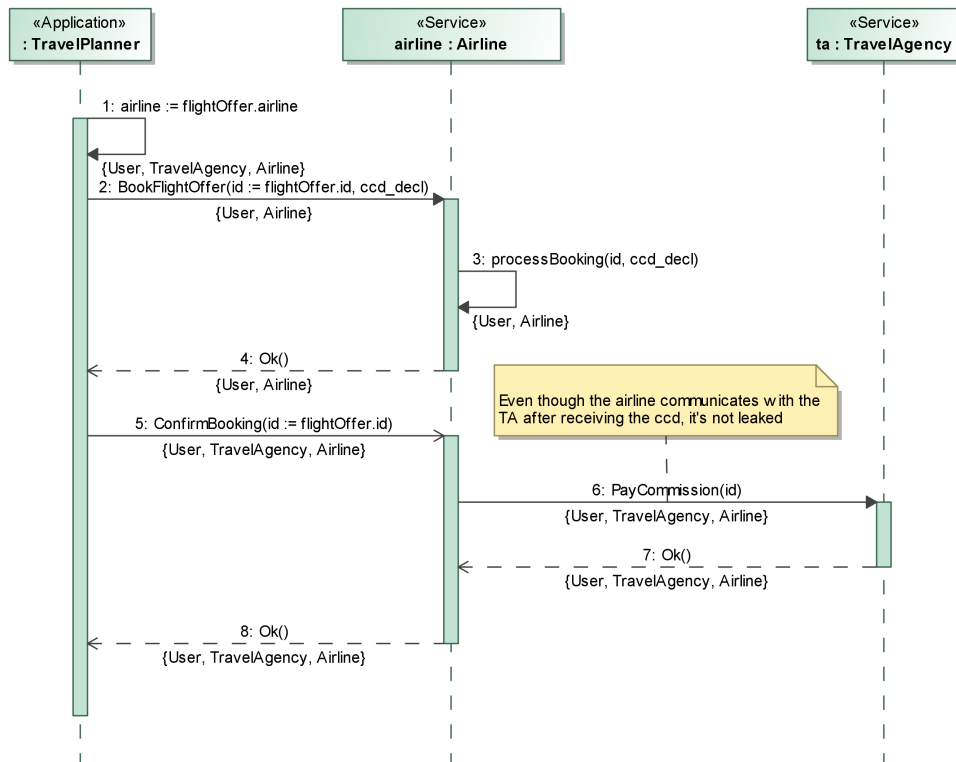


Figure 16: Booking flight

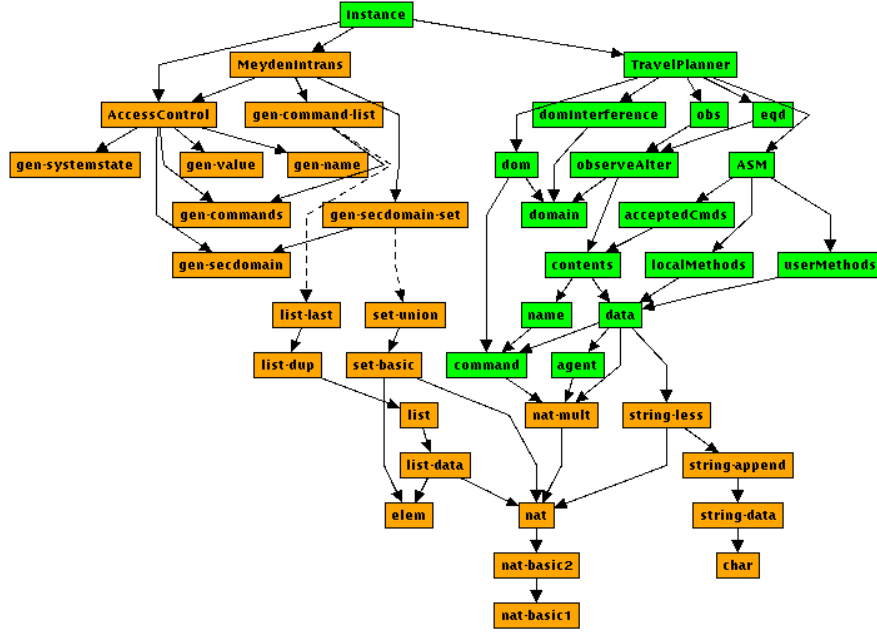


Figure 17: Specification Graph

B KIV specifications

The specification graph of Figure 17 shows the assembly and dependencis of several modular specifications. The color of the nodes indicates different kind of specifications. Orange colored notes show specification which are imported from another KIV project. On the one hand, the specification of the generic noninterference theory of van der Meyden. The single specifications are depicted in Section B.1. On the other hand, some auxiliary specifications are used (see Section B.4. The green nodes are specifications of the current project. The color green indicates that all theorems about these specifications have been proven. These nodes include the system specification of the Travel Planner case study and the formalization of an information flow policy. The specifications are shown in detail in Sections B.2 and B.3 respectively. The specification at the top, called **instance** (see page 42), declares that the Travel Planner instantiates the generic theory and defines the mapping. The instantiation has been proven.

B.1 Specification of generic noninterference theory of van der Meyden

MeydenIntrans, 25
AccessControl, 25
gen-command-list, 26
gen-secdomain-set, 26
gen-commands, 26
gen-name, 26

gen-secdomain, 27
gen-systemstate, 27
gen-value, 27

MeydenIntrans =

enrich AccessControl, gen-command-list, gen-secdomain-set **with**

functions

run : systemstate \times commandlist \rightarrow systemstate ;
run : commandlist \rightarrow systemstate ;
ipurge : commandlist \times secdomain \rightarrow commandlist ;
sources : commandlist \times secdomain \rightarrow secdomainset ;

predicates

eqds : systemstate \times secdomainset \times systemstate;
secure : systemstate;

axioms

eqds-def : eqds(sys, ds, sys₀) \leftrightarrow ($\forall d. d \in ds \rightarrow \text{eqd}(\text{sys}, d, \text{sys}_0)$);
run-base : run(sys, []) = sys;
run-rec : run(sys, co ' + cl) = run(step(sys, co), cl);
run-initial : run(cl) = run(initialstate, cl);
sources-base : sources([], d) = \emptyset ++ d;
sources-rec :
sources(co ' + cl, d)
= ($\exists d_0. d_0 \in \text{sources}(\text{cl}, d) \wedge \text{cdom}(\text{co}) \Rightarrow d_0 \supset \text{sources}(\text{cl}, d) ++$
 $\text{cdom}(\text{co})$; sources(cl, d));
ipurge-base : ipurge([], d) = [];
ipurge-rec :
ipurge(co ' + cl, d)
= ($\text{cdom}(\text{co}) \in \text{sources}(\text{co} + \text{cl}, d) \supset \text{co} + \text{ipurge}(\text{cl}, d)$; ipurge(cl,
d));

end enrich

AccessControl =

enrich gen-secdomain, gen-systemstate, gen-commands, gen-value, gen-name **with**

functions

contents : systemstate \times name \rightarrow value ;
obs : systemstate \times secdomain \rightarrow systemstate ;
step : systemstate \times command \rightarrow systemstate ;
cdom : command \rightarrow secdomain ;

predicates

observe : name \times secdomain;
alter : name \times secdomain;
. \Rightarrow . : secdomain \times secdomain;
eqd : systemstate \times secdomain \times systemstate;

axioms

```

eqd-def :
eqd(sys, d, sys0)  $\leftrightarrow$  ( $\forall$  nam. observe(nam, d)  $\rightarrow$  contents(sys, nam)
= contents(sys0, nam));
obs-in-domain : observe(nam, d)  $\rightarrow$  contents(obs(sys, d), nam) =
contents(sys, nam);
obs-other-domain :  $\neg$  observe(nam, d)  $\rightarrow$  contents(obs(sys, d),
nam) = null;
extensionality : sys = sys0  $\leftrightarrow$  ( $\forall$  nam. contents(sys, nam) =
contents(sys0, nam));
RMA-2 :
  eqd(sys, cdom(co), sys0)
   $\wedge$  contents(sys, nam) = contents(sys0, nam)
   $\wedge$  alter(nam, cdom(co))
 $\rightarrow$  contents(step(sys, co), nam) = contents(step(sys0, co), nam);
RMA-3 : contents(step(sys, co), nam)  $\neq$  contents(sys, nam)  $\rightarrow$ 
alter(nam, cdom(co));
alter-observe-respects-policy : alter(nam, d)  $\wedge$  observe(nam, d0)  $\rightarrow$ 
d  $\Rightarrow$  d0;

```

end enrich

```

gen-command-list =
actualize list-last with gen-commands by morphism
  list  $\rightarrow$  commandlist; elem  $\rightarrow$  command; x  $\rightarrow$  cl; x0  $\rightarrow$  cl0; x1  $\rightarrow$ 
  cl1; y  $\rightarrow$  cl2; z  $\rightarrow$  cl3; y0  $\rightarrow$  cl4; z0  $\rightarrow$  cl5; y1  $\rightarrow$  cl6; z1  $\rightarrow$  cl7;
  x2  $\rightarrow$  cl8; y2  $\rightarrow$  cl9; z2  $\rightarrow$  cl10; a  $\rightarrow$  co; a0  $\rightarrow$  co0; b  $\rightarrow$  co1; c  $\rightarrow$ 
  co2
end actualize

```

```

gen-secdomain-set =
actualize set-union with gen-secdomain by morphism
  set  $\rightarrow$  secdomainset; elem  $\rightarrow$  secdomain; s  $\rightarrow$  ds; s0  $\rightarrow$  ds0; s1
 $\rightarrow$  ds1; s2  $\rightarrow$  ds2; a  $\rightarrow$  d; b  $\rightarrow$  d0; c  $\rightarrow$  d1
end actualize

```

```

gen-commands =
specification
  sorts command;
  variables co: command;
end specification

```

```

gen-name =
specification
  sorts name;
  variables nam: name;
end specification

```

```

gen-secdomain =
specification
  sorts secdomain;
  variables d, d0, d1: secdomain;
end specification

```

```

gen-systemstate =
specification
  sorts systemstate;
  constants initialstate : systemstate;
  variables sys, sys0, sys1: systemstate;
end specification

```

```

gen-value =
specification
  sorts value;
  constants null : value;
  variables v, v0, v1: value;
end specification

```

B.2 Case Study Travel Planner: System Specification

ASM, 27
 prelims-idleState, 38
 localMethods, 39
 userMethods, 39
 contents, 39
 data, 39
 name, 40
 agent, 41
 command, 41

```

ASM =
asm specification ASM
  using localMethods, userMethods, acceptedCmds
  procedures

```

ASM	:	name × agent → data × bool;
STEP	:	name × agent → data × bool;
EXEC	command	: name × agent → data;
START	command	: name × agent → data;
GETINPUTREQUESTDATA	command	: name × agent → data;
RETINPUTREQUESTDATA	command	: name × agent → data;
GETFLIGHTOFFERSTT	command	: name × agent → data;
GETFLIGHTOFFERSTA	command	: name × agent → data;
FILTEROFFERS	command	: name × agent → data;
RETFLIGHTOFFERSAT	command	: name × agent → data;
RETFLIGHTOFFERSTT	command	: name × agent → data;
GETSINGLESELECTION	command	: name × agent → data;
RETSINGLESELECTION	command	: name × agent → data;
RELEASECCD	command	: name × agent → data;
CONFIRMRELEASE	command	: name × agent → data;
RETCONFIRMATION	command	: name × agent → data;
DECLASSIFYCCD	command	: name × agent → data;
DECLASSIFIEDCCD	command	: name × agent → data;
DECLASSYFLIGHTOFFER	command	: name × agent → data;
BOOKFLIGHTOFFER	command	: name × agent → data;
PROCESSBOOKING	command	: name × agent → data;
OKBOOKFLIGHTOFFER	command	: name × agent → data;
SENDCONFIRMBOOKING	command	: name × agent → data;
CONFIRMBOOKING	command	: name × agent → data;
PAYCOMMISSION	command	: name × agent → data;
OKPAYCOMMISSION	command	: name × agent → data;
OKCONFIRMBOOKING	command	: name × agent → data;
PROVIDEINPUT_GETINPUTREQUESTDATA	:	name × agent → data;
PROVIDEINPUT_GETSINGLESELECTION	:	name × agent → data;
PROVIDEINPUT_CONFIRMRELEASE	:	name × agent → data;
U_FINISHED	command	: name × agent → data;
TP_NEXT	command	: name × agent → data;
TA_NEXT	command	: name × agent → data;
A_NEXT	command	: name × agent → data;
CCC_NEXT	command	: name × agent → data;

state variables c, boolvar;
initial state init(c)
final state boolvar
asm rule STEP
declaration

ASM(c, boolvar) { **while** ¬ boolvar **do** STEP }; ;

STEP(c, boolvar) { boolvar := ?;
choose cmd **with** true **in**
 EXEC }; ;

EXEC(cmd; **var** c)
 {
if cmd = StartCmd **then** START **else**

```

        if cmd = GetInputRequestDataCmd then
GETINPUTREQUESTDATA else
        if cmd = RetInputRequestDataCmd then
RETINPUTREQUESTDATA else
        if cmd = GetFlightOffersTTCmd then GETFLIGHTOFFERSTT else
        if cmd = GetFlightOffersTACmd then GETFLIGHTOFFERSTA else
        if cmd = FilterOffersCmd then FILTEROFFERS else
        if cmd = RetFlightOffersATCmd then RETFLIGHTOFFERSAT else
        if cmd = RetFlightOffersTTCmd then RETFLIGHTOFFERSTT else
        if cmd = GetSingleSelectionCmd then GETSINGLESELECTION else
        if cmd = RetSingleSelectionCmd then RETSINGLESELECTION else
        if cmd = ReleaseCCDCmd then RELEASECCD else
        if cmd = ConfirmReleaseCmd then CONFIRMRELEASE else
        if cmd = RetConfirmationCmd then RETCONFIRMATION else
        if cmd = DeclassifyCCDCmd then DECLASSIFYCCD else
        if cmd = DeclassifiedCCDCmd then DECLASSIFIEDCCD else
        if cmd = DeclassifyFlightOfferCmd then DECLASSYFLIGHTOFFER
else
        if cmd = BookFlightOfferCmd then BOOKFLIGHTOFFER else
        if cmd = ProcessBookingCmd then PROCESSBOOKING else
        if cmd = OkBookFlightOfferCmd then OKBOOKFLIGHTOFFER else
        if cmd = SendConfirmBookingCmd then SENDCONFIRMBOOKING
else
        if cmd = ConfirmBookingCmd then CONFIRMBOOKING else
        if cmd = PayCommissionCmd then PAYCOMMISSION else
        if cmd = OkPayCommissionCmd then OKPAYCOMMISSION else
        if cmd = OkConfirmBookingCmd then OKCONFIRMBOOKING
    };

START(cmd; var c)
{
    if TP_receivedExpected(StartCmd, c) then
        {TP_NEXT; c(Inbox(GetInputRequestDataCmd), User(0)) :=
GetInputRequestData}
    };

GETINPUTREQUESTDATA(cmd; var c)
{
    if U_received(GetInputRequestDataCmd, c) then
        {
            PROVIDEINPUT_GETINPUTREQUESTDATA;
            c(Inbox(RetInputRequestDataCmd), TravelPlanner(0)) :=
RetInputRequest-
Data(makeInputRequestData(c(input(GetInputRequestDataCmd),
User(0))));
            U_FINISHED
        }
    };

RETINPUTREQUESTDATA(cmd; var c)

```

```

{
  if TP_receivedExpected(RetInputRequestDataCmd, c) then
  {
    c(requestData, TravelPlanner(0)) :=
c(Inbox(RetInputRequestDataCmd), TravelPlanner(0)).input;
    TP_NEXT;
    c(Inbox(GetFlightOffersTTCmd), TravelAgency(0)) :=
GetFlightOffers(c(requestData, TravelPlanner(0)))
  }
};

GETFLIGHTOFFERSTT(cmd; var c)
{
  if TA_receivedExpected(GetFlightOffersTTCmd, c) then
  {
    c(requestData, TravelAgency(0)) :=
c(Inbox(GetFlightOffersTTCmd), TravelAgency(0)).requestData;
    TA_NEXT;
    c(Inbox(GetFlightOffersTACmd), Airline(0)) :=
GetFlightOffers(c(requestData, TravelAgency(0)))
  }
};

GETFLIGHTOFFERSTA(cmd; var c)
{
  if A_receivedExpected(GetFlightOffersTACmd, c) then
  {
    c(requestData, Airline(0)) := c(Inbox(GetFlightOffersTACmd),
Airline(0)).requestData;
    A_NEXT
  }
};

FILTEROFFERS(cmd; var c)
{
  if c(waiting4, Airline(0)) = Command(FilterOffersCmd) then
  {
    c(filteredFlightOffers, Airline(0)) := filterOffers(c(requestData,
Airline(0)).requestData, c(flightOffers, Airline(0)));
    A_NEXT;
    c(Inbox(RetFlightOffersATCmd), TravelAgency(0)) :=
RetFlightOffers(c(filteredFlightOffers, Airline(0)))
  }
};

RETFLIGHTOFFERSAT(cmd; var c)
{
  if TA_receivedExpected(RetFlightOffersATCmd, c) then
  {

```

```

        c(filteredFlightOffers, TravelAgency(0)) :=
c(Inbox(RetFlightOffersATCmd), TravelAgency(0)).flightOffers;
        TA_NEXT;
        c(Inbox(RetFlightOffersTTCmd), TravelPlanner(0)) :=
RetFlightOffers(c(filteredFlightOffers, TravelAgency(0)))
    }
};

RETFLIGHTOFFERSTT(cmd; var c)
{
if TP_receivedExpected(RetFlightOffersTTCmd, c) then
{
    c(flightOffers, TravelPlanner(0)) :=
c(Inbox(RetFlightOffersTTCmd), TravelPlanner(0)).flightOffers;
    TP_NEXT;
    c(Inbox(GetSingleSelectionCmd), User(0)) :=
GetSingleSelection(c(flightOffers, TravelPlanner(0)))
}
};

GETSINGLESELECTION(cmd; var c)
{
if U_received(GetSingleSelectionCmd, c) then
{
    PROVIDEINPUT_GETSINGLESELECTION;
    c(Inbox(RetSingleSelectionCmd), TravelPlanner(0)) :=
RetSingleSelection(selectOne(c(Inbox(GetSingleSelectionCmd),
User(0)).selection, c(input(GetSingleSelectionCmd), User(0))));
    U_FINISHED
}
};

RETSINGLESELECTION(cmd; var c)
{
if TP_receivedExpected(RetSingleSelectionCmd, c) then
{
    c(flightOffer, TravelPlanner(0)) :=
c(Inbox(RetSingleSelectionCmd), TravelPlanner(0)).selection;
    TP_NEXT;
    c(Inbox(ReleaseCCDCmd), CreditCardCenter(0)) :=
ReleaseCCD(Agent(c(flightOffer, TravelPlanner(0)).airline))
}
};

RELEASECCD(cmd; var c)
{
if CCC_receivedExpected(ReleaseCCDCmd, c) then
{
    c(airline, CreditCardCenter(0)) := c(Inbox(ReleaseCCDCmd),
CreditCardCenter(0)).receiver;

```



```

        CCC_NEXT;
        c(Inbox(ConfirmReleaseCmd), User(0)) := ConfirmRelease(c(ccd,
CreditCardCenter(0)), c(airline, CreditCardCenter(0)))
    }
};

CONFIRMRELEASE(cmd; var c)
{
    if U_received(ConfirmReleaseCmd, c) then
    {
        PROVIDEINPUT_CONFIRMRELEASE;
        if confirmRelease(c(Inbox(ConfirmReleaseCmd), User(0)).what,
c(Inbox(ConfirmReleaseCmd), User(0)).receiver,
c(input(ConfirmReleaseCmd), User(0))).boolean
        then
            c(Inbox(RetConfirmationCmd), CreditCardCenter(0)) :=
RetConfirmation;
            U_FINISHED
    }
};

RETCONFIRMATION(cmd; var c)
{
    if CCC_receivedExpected(RetConfirmationCmd, c) then CCC_NEXT
};

DECLASSIFYCCD(cmd; var c)
{
    if c(waiting4, CreditCardCenter(0)) = Command(DeclassifyCCDCmd)
then
    {
        c(ccd_decl, CreditCardCenter(0)) := declassify(c(ccd,
CreditCardCenter(0)));
        CCC_NEXT;
        c(Inbox(DeclassifiedCCDCmd), TravelPlanner(0)) :=
DeclassifiedCCD(c(ccd_decl, CreditCardCenter(0)))
    }
};

DECLASSIFIEDCCD(cmd; var c)
{
    if TP_receivedExpected(DeclassifiedCCDCmd, c) then
    {
        c(ccd_decl, TravelPlanner(0)) := c(Inbox(DeclassifiedCCDCmd),
TravelPlanner(0)).ccd;
        TP_NEXT
    }
};

DECLASSYFLIGHTOFFER(cmd; var c)

```

```

    {
      if c(waiting4, TravelPlanner(0)) =
Command(DeclassifyFlightOfferCmd) then
        {
          c(id, TravelPlanner(0)) := declassify(c(flightOffer,
TravelPlanner(0)).id);
          TP_NEXT;
          c(Inbox(BookFlightOfferCmd), Airline(0)) :=
BookFlightOffer(c(id, TravelPlanner(0)).id, c(ccd_decl, TravelPlanner(0)))
        }
    };

    BOOKFLIGHTOFFER(cmd; var c)
    {
      if A_receivedExpected(BookFlightOfferCmd, c) then
        {
          c(id, Airline(0)) := c(Inbox(BookFlightOfferCmd), Airline(0));
          c(ccd_decl, Airline(0)) := c(Inbox(BookFlightOfferCmd),
Airline(0)).ccd;
          A_NEXT
        }
    };

    PROCESSBOOKING(cmd; var c)
    {
      if c(waiting4, Airline(0)) = Command(ProcessBookingCmd) then
        {
          if processBooking(c(id, Airline(0)), c(ccd_decl, Airline(0))).boolean
↔ true then
            c(Inbox(OkBookFlightOfferCmd), TravelPlanner(0)) := Ok;
            A_NEXT
          }
    };

    OKBOOKFLIGHTOFFER(cmd; var c)
    {
      if TP_receivedExpected(OkBookFlightOfferCmd, c) then TP_NEXT
    };

    SENDCONFIRMBOOKING(cmd; var c)
    {
      if c(waiting4, TravelPlanner(0)) =
Command(SendConfirmBookingCmd) then
        {
          TP_NEXT;
          c(Inbox(ConfirmBookingCmd), TravelAgency(0)) :=
ConfirmBooking(c(flightOffer, TravelPlanner(0)).id)
        }
    };

    CONFIRMBOOKING(cmd; var c)

```

```

{
  if TA_receivedExpected(ConfirmBookingCmd, c) then
  {
    c(id, TravelAgency(0)) := c(Inbox(ConfirmBookingCmd),
TravelAgency(0)).id;
    TA_NEXT;
    c(Inbox(PayCommissionCmd), Airline(0)) := PayCommission(c(id,
TravelAgency(0)))
  }
};

PAYCOMMISSION(cmd; var c)
{
  if A_receivedExpected(PayCommissionCmd, c) then
  {
    c(id, Airline(0)) := c(Inbox(PayCommissionCmd), Airline(0));
    A_NEXT;
    c(Inbox(OkPayCommissionCmd), TravelAgency(0)) := Ok
  }
};

OKPAYCOMMISSION(cmd; var c)
{
  if TA_receivedExpected(OkPayCommissionCmd, c) then
  {TA_NEXT; c(Inbox(OkConfirmBookingCmd), TravelPlanner(0)) :=
Ok}
};

OKCONFIRMBOOKING(cmd; var c)
{
  if TP_receivedExpected(OkConfirmBookingCmd, c) then TP_NEXT
};

PROVIDEINPUT_GETINPUTREQUESTDATA(c)
{
  c(input(GetInputRequestDataCmd), User(0)) :=
(c(allInputs(GetInputRequestDataCmd),
User(0)).at)(c(inputCounter(GetInputRequestDataCmd), User(0)).nr);
  c(inputCounter(GetInputRequestDataCmd), User(0)) :=
InputCounter(c(inputCounter(GetInputRequestDataCmd), User(0)).nr + 1)
};

PROVIDEINPUT_GETSINGLESELECTION(c)
{
  c(input(GetSingleSelectionCmd), User(0)) :=
(c(allInputs(GetSingleSelectionCmd),
User(0)).at)(c(inputCounter(GetSingleSelectionCmd), User(0)).nr);
  c(inputCounter(GetSingleSelectionCmd), User(0)) :=
InputCounter(c(inputCounter(GetSingleSelectionCmd), User(0)).nr + 1)
};

```

```

    PROVIDEINPUT_CONFIRMRELEASE(c)
    {
        c(input(ConfirmReleaseCmd), User(0)) :=
(c(allInputs(ConfirmReleaseCmd),
User(0)).at(c(inputCounter(ConfirmReleaseCmd), User(0)).nr);
        c(inputCounter(ConfirmReleaseCmd), User(0)) :=
InputCounter(c(inputCounter(ConfirmReleaseCmd), User(0)).nr + 1)
    };

    U_FINISHED(cmd; var c)
    {
        if cmd = GetInputRequestDataCmd then
c(Inbox(GetInputRequestDataCmd), User(0)) := null else
        if cmd = GetSingleSelectionCmd then
            c(Inbox(GetSingleSelectionCmd), User(0)) := null
        else
            if cmd = ConfirmReleaseCmd then c(Inbox(ConfirmReleaseCmd),
User(0)) := null
    };

    TP_NEXT(cmd; var c)
    {
        if cmd = StartCmd then
        {
            c(Inbox(StartCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) :=
Command(RetInputRequestDataCmd)
        }
        else
        if cmd = RetInputRequestDataCmd then
        {
            c(Inbox(RetInputRequestDataCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) :=
Command(RetFlightOffersTTCmd)
        }
        else
        if cmd = RetFlightOffersTTCmd then
        {
            c(Inbox(RetFlightOffersTTCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) :=
Command(RetSingleSelectionCmd)
        }
        else
        if cmd = RetSingleSelectionCmd then
        {
            c(Inbox(RetSingleSelectionCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) := Command(DeclassifiedCCDCmd)
        }
        else
        if cmd = DeclassifiedCCDCmd then

```

```

    {
        c(Inbox(DeclassifiedCCDCmd), TravelPlanner(0)) := null;
        c(waiting4, TravelPlanner(0)) :=
Command(DeclassifyFlightOfferCmd)
    }
    else
    if cmd = DeclassifyFlightOfferCmd then
        c(waiting4, TravelPlanner(0)) := Command(OkBookFlightOfferCmd)
    else
    if cmd = OkBookFlightOfferCmd then
        {
            c(Inbox(OkBookFlightOfferCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) :=
Command(SendConfirmBookingCmd)
        }
    else
    if cmd = SendConfirmBookingCmd then
        c(waiting4, TravelPlanner(0)) := Command(OkConfirmBookingCmd)
    else
    if cmd = OkConfirmBookingCmd then
        {
            c(Inbox(OkConfirmBookingCmd), TravelPlanner(0)) := null;
            c(waiting4, TravelPlanner(0)) := Idle
        }
    };

TA_NEXT(cmd; var c)
{
    if cmd = GetFlightOffersTTCmd then
        {
            c(Inbox(GetFlightOffersTTCmd), TravelAgency(0)) := null;
            c(waiting4, TravelAgency(0)) :=
Command(RetFlightOffersATCmd)
        }
    else
    if cmd = RetFlightOffersATCmd then
        {
            c(Inbox(RetFlightOffersATCmd), TravelAgency(0)) := null;
            c(waiting4, TravelAgency(0)) := Idle
        }
    else
    if cmd = ConfirmBookingCmd then
        {
            c(Inbox(ConfirmBookingCmd), TravelAgency(0)) := null;
            c(waiting4, TravelAgency(0)) :=
Command(OkPayCommissionCmd)
        }
    else
    if cmd = OkPayCommissionCmd then
        {

```

```

                                c(Inbox(OkPayCommissionCmd), TravelAgency(0)) :=
null;
                                c(waiting4, TravelAgency(0)) := Idle
                                }
};

A_NEXT(cmd; var c)
{
  if cmd = GetFlightOffersTACmd then
  {
    c(Inbox(GetFlightOffersTACmd), Airline(0)) := null;
    c(waiting4, Airline(0)) := Command(FilterOffersCmd)
  }
  else
  if cmd = FilterOffersCmd then
    c(waiting4, Airline(0)) := Command(GetFlightOffersTACmd)
  else
  if cmd = BookFlightOfferCmd then
  {
    c(Inbox(BookFlightOfferCmd), Airline(0)) := null;
    c(waiting4, Airline(0)) := Command(ProcessBookingCmd)
  }
  else
  if cmd = ProcessBookingCmd then c(waiting4, Airline(0)) :=
Idle else
    if cmd = PayCommissionCmd then
    {
      c(Inbox(PayCommissionCmd), Airline(0)) := null;
      c(waiting4, Airline(0)) := Idle
    }
};

CCC_NEXT(cmd; var c)
{
  if cmd = ReleaseCCDCmd then
  {
    c(Inbox(ReleaseCCDCmd), CreditCardCenter(0)) := null;
    c(waiting4, CreditCardCenter(0)) :=
Command(RetConfirmationCmd)
  }
  else
  if cmd = RetConfirmationCmd then
  {
    c(Inbox(RetConfirmationCmd), CreditCardCenter(0)) := null;
    c(waiting4, CreditCardCenter(0)) :=
Command(DeclassifyCCDCmd)
  }
  else
  if cmd = DeclassifyCCDCmd then c(waiting4,
CreditCardCenter(0)) := Idle

```

} end asm specification

prelims-idleState =

enrich localMethods, userMethods, contents **with**

predicates

init	:	(name \times agent \rightarrow data);
U_received	:	command \times (name \times agent \rightarrow data);
TP_receivedExpected	:	command \times (name \times agent \rightarrow data);
TA_receivedExpected	:	command \times (name \times agent \rightarrow data);
A_receivedExpected	:	command \times (name \times agent \rightarrow data);
CCC_receivedExpected	:	command \times (name \times agent \rightarrow data);
TP_acceptedFromIdle	:	command;
TA_acceptedFromIdle	:	command;
A_acceptedFromIdle	:	command;
CCC_acceptedFromIdle	:	command;

axioms

init-def :

init(c)

\leftrightarrow TP_receivedExpected(StartCmd, c)

\wedge c(waiting4, TravelAgency(0)) = Idle

\wedge c(waiting4, Airline(0)) = Idle

\wedge c(waiting4, CreditCardCenter(0)) = Idle;

U_received-def : U_received(cmd, c) \leftrightarrow c(Inbox(cmd), User(0)) \neq null;

TP_receivedExpected-def :

TP_receivedExpected(cmd, c)

\leftrightarrow (TP_acceptedFromIdle(cmd) \rightarrow c(waiting4, TravelPlanner(0)) = Idle)

\wedge (\neg TP_acceptedFromIdle(cmd) \rightarrow c(waiting4,

TravelPlanner(0)) = Command(cmd))

\wedge c(Inbox(cmd), TravelPlanner(0)) \neq null;

TA_receivedExpected-def :

TA_receivedExpected(cmd, c)

\leftrightarrow (TA_acceptedFromIdle(cmd) \rightarrow c(waiting4, TravelAgency(0)) = Idle)

\wedge (\neg TA_acceptedFromIdle(cmd) \rightarrow c(waiting4,

TravelAgency(0)) = Command(cmd))

\wedge c(Inbox(cmd), TravelAgency(0)) \neq null;

A_receivedExpected-def :

A_receivedExpected(cmd, c)

\leftrightarrow (A_acceptedFromIdle(cmd) \rightarrow c(waiting4, Airline(0)) = Idle)

\wedge (\neg A_acceptedFromIdle(cmd) \rightarrow c(waiting4, Airline(0)) =

Command(cmd))

\wedge c(Inbox(cmd), Airline(0)) \neq null;

CCC_receivedExpected-def :

CCC_receivedExpected(cmd, c)

```

 $\leftrightarrow$  (CCC_acceptedFromIdle(cmd)  $\rightarrow$  c(waiting4,
CreditCardCenter(0)) = Idle)
 $\wedge$  ( $\neg$  CCC_acceptedFromIdle(cmd)  $\rightarrow$  c(waiting4,
CreditCardCenter(0)) = Command(cmd))
 $\wedge$  c(Inbox(cmd), CreditCardCenter(0))  $\neq$  null;
TP_acceptedFromIdle-def : TP_acceptedFromIdle(cmd)  $\leftrightarrow$  cmd =
StartCmd;
TA_acceptedFromIdle-def :
TA_acceptedFromIdle(cmd)  $\leftrightarrow$  cmd = GetFlightOffersTTCmd  $\vee$ 
cmd = ConfirmBookingCmd;
A_acceptedFromIdle-def :
A_acceptedFromIdle(cmd)  $\leftrightarrow$  cmd = GetFlightOffersTACmd  $\vee$ 
cmd = BookFlightOfferCmd;
CCC_acceptedFromIdle-def : CCC_acceptedFromIdle(cmd)  $\leftrightarrow$  cmd
= ReleaseCCDCmd;

```

end enrich

```

localMethods =
enrich data with
  functions
    declassify      : data           $\rightarrow$  data ;
    filterOffers    : data  $\times$  data  $\rightarrow$  data ;
    processBooking  : data  $\times$  data  $\rightarrow$  data ;

```

end enrich

```

userMethods =
enrich data with
  functions
    makeInputRequestData : data           $\rightarrow$  data ;
    selectOne             : data  $\times$  data  $\rightarrow$  data ;
    selectMultiple        : data  $\times$  data  $\rightarrow$  data ;
    confirmRelease        : data  $\times$  data  $\times$  data  $\rightarrow$  data ;

```

end enrich

```

contents =
enrich name, data with
  variables c, c0, c1, c2: name  $\times$  agent  $\rightarrow$  data;

```

end enrich

```

data =
data specification
  using command, string-less, nat-mult, agent

```



```

data = CreditCardDetails (. .data : string ;) with isCreditCardDetails
| FlightOffer (. .id : data ; . .airline : agent ;) with isFlightOffer
| RequestData (. .data : string ;) with isRequestData
| GetFlightOffers (. .requestData : data ;) with isGetFlightOffers
| RetFlightOffers (. .flightOffers : data ;) with isRetFlightOffers
| ReleaseCCD (. .receiver : data ;) with isReleaseCCD
| DeclassifiedCCD (. .ccd : data ;) with isDeclassifiedCCD
| BookFlightOffer (. .id : data ; . .ccd : data ;) with isBookFlightOffer
| ConfirmBooking (. .id : data ;)
| PayCommission (. .id : data ;)
| DataList (. .list : dataList ;) with isDataList
| Input (. .at : nat → data ;) with isInput
| InputCounter (. .nr : nat ;) with isInputCounter
| Boolean (. .boolean : bool ;) with isBoolean
| Number (. .number : nat ;) with isNumber
| Command (. .cmd : command ;) with isCommand
| Idle
| Agent (. .agent : agent ;) with isAgent
| null
| Start
| GetInputRequestData
| RetInputRequestData (. .input : data ;) with isRetInputRequestData
| GetSingleSelection (. .selection : data ;) with isGetSingleSelection
| RetSingleSelection (. .selection : data ;) with isRetSingleSelection
| GetMultipleSelection (. .selection : data ;) with isGetMultipleSelection
| RetMultipleSelection (. .selection : data ;) with isRetMultipleSelection
| ConfirmRelease (. .what : data ; . .receiver : data ;) with isConfirmRelease
| RetConfirmation
| Show (. .data : data ;) with isShow
| Ok
;
dataList = []
| . + . prio 9 (. .first : data ; . .rest : dataList ;) prio 9
;
variables
d, d0, d1: data;
dl, dl0, dl1: dataList;
userInputs, userInputs0, userInputs1: nat → data;
end data specification

```

```

name =
data specification
using command
name = ccd
| ccd_decl
| flightOffer
| flightOffer_decl
| requestData

```

```

| flightOffers
| filteredFlightOffers
| airline
| id
| allInputs (. .cmd : command ;) with isAllInputs
| inputCounter (. .cmd : command ;) with isInputCounter
| input (. .cmd : command ;) with isInput
| waiting4
| Inbox (. .cmd : command ;) with isInbox
;
variables nam, nam0, nam1: name;
end data specification

```

```

agent =
data specification
using nat-mult
agent = User (. .id : nat ;) with isUser
| TravelPlanner (. .id : nat ;) with isTravelPlanner
| TravelAgency (. .id : nat ;) with isTravelAgency
| Airline (. .id : nat ;) with isAirline
| CreditCardCenter (. .id : nat ;) with isCreditCardCenter
;
variables ag, ag0, ag1: agent;
end data specification

```

```

command =
data specification
using nat-mult
command = StartCmd
| GetInputRequestDataCmd
| RetInputRequestDataCmd
| GetFlightOffersTTCmd
| GetFlightOffersTACmd
| FilterOffersCmd
| RetFlightOffersATCmd
| RetFlightOffersTTCmd
| GetSingleSelectionCmd
| RetSingleSelectionCmd
| ReleaseCCDCmd
| ConfirmReleaseCmd
| RetConfirmationCmd
| DeclassifyCCDCmd
| DeclassifiedCCDCmd
| DeclassifyFlightOfferCmd
| BookFlightOfferCmd
| ProcessBookingCmd
| OkBookFlightOfferCmd

```

```

| SendConfirmBookingCmd
| ConfirmBookingCmd
| PayCommissionCmd
| OkPayCommissionCmd
| OkConfirmBookingCmd
| GetMultipleSelectionCmd
| RetMultipleSelectionCmd
| Show
;
variables cmd, cmd0, cmd1, cmd2: command;
end data specification

```

B.3 Case Study Travel Planner: Information Flow Property Specification

Instance, 42
Security, 42
dom, 43
domInterference, 44
eqd, 44
obs, 44
observeAlter, 45
domain, 48
Instance =
instantiate (AccessControl) < MeydenIntrans **with** TravelPlanner **by**
mapping
systemstate \rightarrow (name \times agent \rightarrow data); secdomain \rightarrow domain;
value \rightarrow data; name \rightarrow name, agent; initialstate \rightarrow (λ nam, ag.
null); null \rightarrow (null); cdom \rightarrow (dom); contents \rightarrow (λ c, nam, ag.
c(nam, ag)); step \rightarrow (STEP#); obs \rightarrow (obs); observe \rightarrow (λ nam,
ag, domvar. observe(domvar, nam, ag)); alter \rightarrow (λ nam, ag,
domvar. alter(domvar, nam, ag)); $\Rightarrow \rightarrow$ (\Rightarrow); eqd \rightarrow (eqd); sys
 \rightarrow c; sys₀ \rightarrow c₀; sys₁ \rightarrow c₁; d \rightarrow domvar; d₀ \rightarrow domvar₀; d₁ \rightarrow
domvar₁; nam \rightarrow nam, ag; v \rightarrow v; v₀ \rightarrow v₀; v₁ \rightarrow v₁ **rename**
end instantiate

Security =
enrich ASM, eqd, dom, domInterference, obs **with**
procedures STEP#name \times agent \rightarrow data \times command : name \times agent \rightarrow data;
declaration
STEP#(c, cmd; **var** c₀) { EXEC(cmd; c);
c₀ := c };
axioms
User_TA_A_Dom \Rightarrow User_A_Dom;
User_TA_A_Dom \Rightarrow User_Decl2A_Dom;

$\text{User_A_Dom} \Rightarrow \text{User_Decl2A_Dom};$
 $\text{User_Decl2A_Dom} \Rightarrow \text{Decl2A_Dom};$
 $\text{Decl2A_Dom} \Rightarrow \text{User_A_Dom};$
 $\neg \text{User_A_Dom} \Rightarrow \text{User_TA_A_Dom};$
 $\neg \text{User_Decl2A_Dom} \Rightarrow \text{User_TA_A_Dom};$
 $\neg \text{Decl2A_Dom} \Rightarrow \text{User_TA_A_Dom};$
 $\neg \text{User_Decl2A_Dom} \Rightarrow \text{User_A_Dom};$
 $\neg \text{Decl2A_Dom} \Rightarrow \text{User_Decl2A_Dom};$
 $\neg \text{User_A_Dom} \Rightarrow \text{Decl2A_Dom};$
 $\neg \text{User_TA_A_Dom} \Rightarrow \text{Decl2A_Dom};$

end enrich

$\text{dom} =$
enrich domain, command **with**
functions $\text{dom} : \text{command} \rightarrow \text{domain} ;$

axioms

$\text{dom}(\text{StartCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{GetInputRequestDataCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{RetInputRequestDataCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{GetFlightOffersTTCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{GetFlightOffersTACmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{FilterOffersCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{RetFlightOffersATCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{RetFlightOffersTTCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{GetSingleSelectionCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{RetSingleSelectionCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{ReleaseCCDCmd}) = \text{User_Decl2A_Dom};$
 $\text{dom}(\text{ConfirmReleaseCmd}) = \text{User_Decl2A_Dom};$
 $\text{dom}(\text{RetConfirmationCmd}) = \text{User_Decl2A_Dom};$
 $\text{dom}(\text{DeclassifyCCDCmd}) = \text{Decl2A_Dom};$
 $\text{dom}(\text{DeclassifiedCCDCmd}) = \text{User_A_Dom};$
 $\text{dom}(\text{DeclassifyFlightOfferCmd}) = \text{User_A_Dom};$
 $\text{dom}(\text{BookFlightOfferCmd}) = \text{User_A_Dom};$
 $\text{dom}(\text{ProcessBookingCmd}) = \text{User_A_Dom};$
 $\text{dom}(\text{OkBookFlightOfferCmd}) = \text{User_A_Dom};$
 $\text{dom}(\text{SendConfirmBookingCmd}) = \text{User_TA_A_Dom};$
 $\text{dom}(\text{ConfirmBookingCmd}) = \text{User_TA_A_Dom};$

```

dom(PayCommissionCmd) = User_TA_A_Dom;
dom(OkPayCommissionCmd) = User_TA_A_Dom;
dom(OkConfirmBookingCmd) = User_TA_A_Dom;

```

end enrich

```

domInterference =
enrich domain with
  predicates .  $\Rightarrow$  . : domain  $\times$  domain;

```

axioms

```

reflexivity : domvar  $\Rightarrow$  domvar;
User_TA_A_Dom  $\Rightarrow$  User_A_Dom;
User_TA_A_Dom  $\Rightarrow$  User_Decl2A_Dom;
User_A_Dom  $\Rightarrow$  User_Decl2A_Dom;
User_Decl2A_Dom  $\Rightarrow$  Decl2A_Dom;
Decl2A_Dom  $\Rightarrow$  User_A_Dom;
Decl2A_Dom  $\Rightarrow$  User_Decl2A_Dom;
User_A_Dom  $\Rightarrow$  User_TA_A_Dom;

```

end enrich

```

eqd =
enrich observeAlter with
  predicates eqd : (name  $\times$  agent  $\rightarrow$  data)  $\times$  domain  $\times$  (name  $\times$  agent  $\rightarrow$  data);

```

axioms

```

eqd :
eqd(c, domvar, c0)  $\leftrightarrow$  ( $\forall$  nam, ag. observe(domvar, nam, ag)  $\rightarrow$ 
c(nam, ag) = c0(nam, ag));

```

end enrich

```

obs =
enrich observeAlter with
  functions obs : (name  $\times$  agent  $\rightarrow$  data)  $\times$  domain  $\rightarrow$  name  $\times$  agent  $\rightarrow$  data ;

```

axioms

```

obs-in-domain : observe(domvar, nam, ag)  $\rightarrow$  (obs(c,
domvar))(nam, ag) = c(nam, ag);
obs-other-domain :  $\neg$  observe(domvar, nam, ag)  $\rightarrow$  (obs(c,
domvar))(nam, ag) = null;

```

end enrich

observeAlter =

enrich domain, contents **with**

predicates

observe : domain \times name \times agent;
alter : domain \times name \times agent;

axioms

observe-User_TA_A_Dom :
 observe(User_TA_A_Dom, nam, ag)
 $\leftrightarrow \neg \neg ($ nam = waiting4 \wedge ag = TravelPlanner(0)
 \vee nam = Inbox(StartCmd) \wedge ag = TravelPlanner(0)
 \vee nam = Inbox(GetInputRequestDataCmd) \wedge ag =
User(0)
 \vee nam = inputCounter(GetInputRequestDataCmd) \wedge ag =
= User(0)
 \vee nam = allInputs(GetInputRequestDataCmd) \wedge ag =
User(0)
 \vee nam = input(GetInputRequestDataCmd) \wedge ag =
User(0)
 \vee nam = inputCounter(GetSingleSelectionCmd) \wedge ag =
User(0)
 \vee nam = allInputs(GetSingleSelectionCmd) \wedge ag =
User(0)
 \vee nam = input(GetSingleSelectionCmd) \wedge ag = User(0)
 \vee nam = Inbox(RetInputRequestDataCmd) \wedge ag =
TravelPlanner(0)
 \vee nam = requestData \wedge ag = TravelPlanner(0)
 \vee nam = Inbox(GetFlightOffersTTCmd) \wedge ag =
TravelAgency(0)
 \vee nam = waiting4 \wedge ag = TravelAgency(0)
 \vee nam = requestData \wedge ag = TravelAgency(0)
 \vee nam = Inbox(GetFlightOffersTACmd) \wedge ag = Airline(0)
 \vee nam = waiting4 \wedge ag = Airline(0)
 \vee nam = Inbox(GetFlightOffersTACmd) \wedge ag = Airline(0)
 \vee nam = requestData \wedge ag = Airline(0)
 \vee nam = flightOffers \wedge ag = Airline(0)
 \vee nam = filteredFlightOffers \wedge ag = Airline(0)
 \vee nam = Inbox(RetFlightOffersATCmd) \wedge ag =
TravelAgency(0)
 \vee nam = Inbox(RetFlightOffersTTCmd) \wedge ag =
TravelPlanner(0)
 \vee nam = flightOffers \wedge ag = TravelPlanner(0)
 \vee nam = Inbox(GetSingleSelectionCmd) \wedge ag = User(0)
 \vee nam = Inbox(RetSingleSelectionCmd) \wedge ag =
TravelPlanner(0)
 \vee nam = flightOffer \wedge ag = TravelPlanner(0)

$\vee \text{nam} = \text{Inbox}(\text{ConfirmBookingCmd}) \wedge \text{ag} =$
 $\text{TravelAgency}(0)$
 $\vee \text{nam} = \text{id} \wedge \text{ag} = \text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{PayCommissionCmd}) \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{OkPayCommissionCmd}) \wedge \text{ag} =$
 $\text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{OkConfirmBookingCmd}) \wedge \text{ag} =$
 $\text{TravelPlanner}(0));$
 $\text{alter-User_TA_A_Dom} :$
 $\text{alter}(\text{User_TA_A_Dom}, \text{nam}, \text{ag})$
 $\leftrightarrow \neg \neg (\vee \text{nam} = \text{waiting4} \wedge \text{ag} = \text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{StartCmd}) \wedge \text{ag} = \text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{inputCounter}(\text{GetInputRequestDataCmd}) \wedge \text{ag} =$
 $= \text{User}(0)$
 $\vee \text{nam} = \text{input}(\text{GetInputRequestDataCmd}) \wedge \text{ag} =$
 $\text{User}(0)$
 $\vee \text{nam} = \text{inputCounter}(\text{GetSingleSelectionCmd}) \wedge \text{ag} =$
 $\text{User}(0)$
 $\vee \text{nam} = \text{input}(\text{GetSingleSelectionCmd}) \wedge \text{ag} = \text{User}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{RetInputRequestDataCmd}) \wedge \text{ag} =$
 $\text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{GetInputRequestDataCmd}) \wedge \text{ag} =$
 $\text{User}(0)$
 $\vee \text{nam} = \text{requestData} \wedge \text{ag} = \text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{GetFlightOffersTTCmd}) \wedge \text{ag} =$
 $\text{TravelAgency}(0)$
 $\vee \text{nam} = \text{waiting4} \wedge \text{ag} = \text{TravelAgency}(0)$
 $\vee \text{nam} = \text{requestData} \wedge \text{ag} = \text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{GetFlightOffersTACmd}) \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{waiting4} \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{GetFlightOffersTACmd}) \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{requestData} \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{filteredFlightOffers} \wedge \text{ag} = \text{Airline}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{RetFlightOffersATCmd}) \wedge \text{ag} =$
 $\text{TravelAgency}(0)$
 $\vee \text{nam} = \text{filteredFlightOffers} \wedge \text{ag} = \text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{RetFlightOffersTTCmd}) \wedge \text{ag} =$
 $\text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{flightOffers} \wedge \text{ag} = \text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{GetSingleSelectionCmd}) \wedge \text{ag} = \text{User}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{RetSingleSelectionCmd}) \wedge \text{ag} =$
 $\text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{flightOffer} \wedge \text{ag} = \text{TravelPlanner}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{ReleaseCCDCmd}) \wedge \text{ag} =$
 $\text{CreditCardCenter}(0)$
 $\vee \text{nam} = \text{id} \wedge \text{ag} = \text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{ConfirmBookingCmd}) \wedge \text{ag} =$
 $\text{TravelAgency}(0)$
 $\vee \text{nam} = \text{Inbox}(\text{PayCommissionCmd}) \wedge \text{ag} = \text{Airline}(0)$

```

    ∨ nam = id ∧ ag = Airline(0)
    ∨ nam = Inbox(OkPayCommissionCmd) ∧ ag =
TravelAgency(0)
    ∨ nam = Inbox(OkConfirmBookingCmd) ∧ ag =
TravelPlanner(0));
observe-User_Decl2A_Dom :
    observe(User_Decl2A_Dom, nam, ag)
↔ ¬ ¬ (  nam = Inbox(ReleaseCCDCmd) ∧ ag =
CreditCardCenter(0)
    ∨ nam = waiting4 ∧ ag = CreditCardCenter(0)
    ∨ nam = airline ∧ ag = CreditCardCenter(0)
    ∨ nam = ccd ∧ ag = CreditCardCenter(0)
    ∨ nam = Inbox(ConfirmReleaseCmd) ∧ ag = User(0)
    ∨ nam = inputCounter(ConfirmReleaseCmd) ∧ ag =
User(0)
    ∨ nam = allInputs(ConfirmReleaseCmd) ∧ ag = User(0)
    ∨ nam = input(ConfirmReleaseCmd) ∧ ag = User(0)
    ∨ nam = Inbox(RetConfirmationCmd) ∧ ag =
CreditCardCenter(0));
alter-User_Decl2A_Dom :
    alter(User_Decl2A_Dom, nam, ag)
↔ ¬ ¬ (  nam = Inbox(ReleaseCCDCmd) ∧ ag =
CreditCardCenter(0)
    ∨ nam = waiting4 ∧ ag = CreditCardCenter(0)
    ∨ nam = airline ∧ ag = CreditCardCenter(0)
    ∨ nam = Inbox(ConfirmReleaseCmd) ∧ ag = User(0)
    ∨ nam = input(ConfirmReleaseCmd) ∧ ag = User(0)
    ∨ nam = inputCounter(ConfirmReleaseCmd) ∧ ag =
User(0)
    ∨ nam = Inbox(RetConfirmationCmd) ∧ ag =
CreditCardCenter(0));
observe-Decl2A_Dom :
    observe(Decl2A_Dom, nam, ag)
↔ ¬ ¬ (  nam = waiting4 ∧ ag = CreditCardCenter(0)
    ∨ nam = ccd ∧ ag = CreditCardCenter(0)
    ∨ nam = ccd_decl ∧ ag = CreditCardCenter(0));
alter-Decl2A_Dom :
    alter(Decl2A_Dom, nam, ag)
↔ ¬ ¬ (  nam = waiting4 ∧ ag = CreditCardCenter(0)
    ∨ nam = ccd_decl ∧ ag = CreditCardCenter(0)
    ∨ nam = Inbox(DeclassifiedCCDCmd) ∧ ag =
TravelPlanner(0));
observe-User_A_Dom :
    observe(User_A_Dom, nam, ag)
↔ ¬ ¬ (  nam = waiting4 ∧ ag = TravelPlanner(0)
    ∨ nam = ccd_decl ∧ ag = CreditCardCenter(0)
    ∨ nam = Inbox(DeclassifiedCCDCmd) ∧ ag =
TravelPlanner(0)

```



```

    ∨ nam = flightOffer ∧ ag = TravelPlanner(0)
    ∨ nam = id ∧ ag = TravelPlanner(0)
    ∨ nam = ccd_decl ∧ ag = TravelPlanner(0)
    ∨ nam = Inbox(BookFlightOfferCmd) ∧ ag = Airline(0)
    ∨ nam = waiting4 ∧ ag = Airline(0)
    ∨ nam = id ∧ ag = Airline(0)
    ∨ nam = ccd_decl ∧ ag = Airline(0)
    ∨ nam = Inbox(OkBookFlightOfferCmd) ∧ ag =
TravelPlanner(0));
alter-User_A_Dom :
    alter(User_A_Dom, nam, ag)
↔ ¬ ¬ ¬ (   nam = waiting4 ∧ ag = TravelPlanner(0)
           ∨ nam = Inbox(DeclassifiedCCDCmd) ∧ ag =
TravelPlanner(0)
           ∨ nam = ccd_decl ∧ ag = TravelPlanner(0)
           ∨ nam = id ∧ ag = TravelPlanner(0)
           ∨ nam = Inbox(BookFlightOfferCmd) ∧ ag = Airline(0)
           ∨ nam = waiting4 ∧ ag = Airline(0)
           ∨ nam = id ∧ ag = Airline(0)
           ∨ nam = ccd_decl ∧ ag = Airline(0)
           ∨ nam = Inbox(OkBookFlightOfferCmd) ∧ ag =
TravelPlanner(0));

```

end enrich

```

domain =
data specification
    domain = User_Decl2A_Dom
           | User_A_Dom
           | User_TA_A_Dom
           | Decl2A_Dom
           ;
    variables domvar, domvar0, domvar1: domain;
end data specification

```

B.4 Auxiliary Specifications

```

list-last, 49
set-union, 49
list-dup, 50
set-basic, 50
list, 51
list-data, 51
string-less, 51
elem, 52
nat-mult, 53
string-append, 53

```

nat, 53
 string-data, 54
 char, 54
 nat-basic2, 54
 nat-basic1, 54

list-last =

enrich list-dup **with**

functions

$.last : list \rightarrow elem$;
 $.butlast : list \rightarrow list$;
 $butlastn : nat \times list \rightarrow list$;
 $rev : list \rightarrow list$;
 $mklist : elem \times nat \rightarrow list$;
 $fillfirst : nat \times elem \times list \rightarrow list$;

predicates

$. \sqsubseteq . : list \times list$;
 $. \sqsupseteq . : list \times list$;

axioms

$laststep : x \neq [] \rightarrow x.butlast + x.last = x$;
 $rev-e : rev([]) = []$;
 $rev-r : rev(a' + x) = rev(x) + a$;
 $mk-len : \# mklist(a, n) = n$;
 $mk-elem : a \in mklist(b, n) \rightarrow a = b$;
 $butlastN-base : butlastn(0, x) = x$;
 $butlastN-rec : butlastn(n + 1, x) = butlastn(n, x.butlast)$;
 $fillfirst-longer : n \leq \# x \rightarrow fillfirst(n, a, x) = x$;
 $fillfirst-fill : \# x < n \rightarrow fillfirst(n, a, x) = mklist(a, n - \# x) + x$;
 $prefix : x \sqsubseteq y \leftrightarrow (\exists z. x + z = y)$;
 $postfix : x \sqsupseteq y \leftrightarrow (\exists z. z + x = y)$;

end enrich

set-union =

enrich set-basic **with**

functions

$\{ . \} : elem \rightarrow set$;
 $. \cup . : set \times set \rightarrow set$ **prio 9** ;
 $. \cap . : set \times set \rightarrow set$ **prio 9** ;
 $. \setminus . : set \times set \rightarrow set$ **prio 9** ;
 $. - . : set \times elem \rightarrow set$ **prio 9 left** ;

axioms

One : $\{a\} = \emptyset ++ a$;
 Union : $a \in s_1 \cup s_2 \leftrightarrow a \in s_1 \vee a \in s_2$;
 Intersect : $a \in s_1 \cap s_2 \leftrightarrow a \in s_1 \wedge a \in s_2$;
 Difference : $a \in s_1 \setminus s_2 \leftrightarrow a \in s_1 \wedge \neg a \in s_2$;
 Delete : $a \in s - b \leftrightarrow a \neq b \wedge a \in s$;

end enrich

list-dup =
enrich list with
 functions
 . ++ . : list \times elem \rightarrow list **prio 9 left**;
 rmdup : list \rightarrow list ;
 predicates
 dups : list;
 disj : list \times list;

axioms

 rmdup-e : rmdup([]) = [];
 rmdup-y : $a \in x \rightarrow \text{rmdup}(a ' + x) = \text{rmdup}(x)$;
 rmdup-n : $\neg a \in x \rightarrow \text{rmdup}(a ' + x) = a + \text{rmdup}(x)$;
 adjoin-in : $a \in x \rightarrow x ++ a = x$;
 adjoin-notin : $\neg a \in x \rightarrow x ++ a = a + x$;
 dups : $\text{dups}(x) \leftrightarrow (\exists a, x_0, y, z. x = x_0 + a + y + a + z)$;
 disjoint : $\text{disj}(x, y) \leftrightarrow (\forall a. \neg (a \in x \wedge a \in y))$;

end enrich

set-basic =
generic specification
 parameter elem **using** nat **target**
 sorts set;
 constants \emptyset : set;
 functions
 . ++ . : set \times elem \rightarrow set **prio 9 left**;
 # . : set \rightarrow nat ;
 predicates
 . \in . : elem \times set;
 . \subseteq . : set \times set;
 variables s, s₀, s₁, s₂ : set;

axioms

```

set generated by  $\emptyset, ++$ ;
Extension :  $s_1 = s_2 \leftrightarrow (\forall a. a \in s_1 \leftrightarrow a \in s_2)$ ;
In-empty :  $\neg a \in \emptyset$ ;
In-insert :  $a \in s ++ b \leftrightarrow a = b \vee a \in s$ ;
Size-empty :  $\# \emptyset = 0$ ;
Size-insert :  $\neg a \in s \rightarrow \#(s ++ a) = \# s + 1$ ;
Subset :  $s_1 \subseteq s_2 \leftrightarrow (\forall a. a \in s_1 \rightarrow a \in s_2)$ ;

```

end generic specification

```

list =
enrich list-data with
  functions
    . '      : elem      → list  ;
    . + .    : list × list → list prio 9;
    . + .    : list × elem → list prio 9;
    . + .    : elem × elem → list prio 9;
  predicates . ∈ . : elem × list;

```

axioms

```

Nil :  $[] + x = x$ ;
Cons :  $(a + x) + y = a + x + y$ ;
One :  $a ' = a + []$ ;
Last :  $x + a = x + a '$ ;
Two :  $a + b = a ' + b '$ ;
In :  $a \in x \leftrightarrow (\exists y, z. x = y + a + z)$ ;

```

end enrich

```

list-data =
generic data specification
  parameter elem using nat
  list = []
    | . + . prio 9 ( . .first : elem ; . .rest : list ; ) prio 9
    ;
  variables x, y, z, x0, y0, z0, x1, y1, z1, x2, y2, z2: list;
  size functions # . : list → nat ;
  order predicates . < . : list × list;
end generic data specification

```

```

string-less =
enrich string-append, nat with
  constants sortedchars : string;

```

functions

. : string → nat ;
 pos : char × string → nat ;

predicates

. < . : string × string;
 . < . : char × char;

variables char₃: char;

axioms

Length-empty : # "" = 0;
 Length-rec : #(char ' + str) = # str + 1;
 Pos-empty : pos(char, "") = 0;
 Pos-found : pos(char, char ' + str) = 0;
 Pos-rec : char₁ ≠ char₂ → pos(char₁, char₂ ' + str) = pos(char₁, str) + 1;
 Irreflexivity : ¬ str < str;
 Transitivity : str₁ < str₂ ∧ str₂ < str₃ → str₁ < str₃;
 Totality : str₁ < str₂ ∨ str₁ = str₂ ∨ str₂ < str₁;
 Less-empty : "" < char ' + str;
 Less-notempty : ¬ str < "";
 Less-rec : char₁ ' + str₁ < char₂ ' + str₂ ↔ char₁ < char₂ ∨ char₁ = char₂ ∧ str₁ < str₂;
 Char-irref : ¬ char < char;
 Char-trans : char₁ < char₂ ∧ char₂ < char₃ → char₁ < char₃;
 Char-total : char₁ < char₂ ∨ char₁ = char₂ ∨ char₂ < char₁;
 Char-less : char₁ < char₂ ↔ pos(char₁, sortedchars) < pos(char₂, sortedchars);
 Sortedchars :
 sortedchars
 = "!#\$%&()*+-. /0123456789:<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[]^_abcdefghijklmnopqrstuvwxyz|~";

end enrich

elem =

specification

sorts elem;
variables a, b, c: elem;

end specification

```

nat-mult =
enrich nat with
  functions . * . : nat × nat → nat prio 10;

```

axioms

```

  m * 0 = 0;
  m * n + 1 = m * n + m;

```

end enrich

```

string-append =
enrich string-data with
  functions
    . + . : string × string → string prio 9;
    . ' : char → string ;
  variables stringvar, stringvar0: string;

```

axioms

```

  chartostring : char ' = char + “”;
  append-base : “” + str = str;
  append-rec : (char + str) + str0 = char + str + str0;

```

end enrich

```

nat =
enrich nat-basic2 with
  functions . - . : nat × nat → nat prio 8 left;
  predicates
    . ≤ . : nat × nat;
    . > . : nat × nat;
    . ≥ . : nat × nat;

```

axioms

```

  m - 0 = m;
  m - n + 1 = (m - n)-1;
  m ≤ n ↔ ¬ n < m;
  m > n ↔ n < m;
  m ≥ n ↔ ¬ m < n;

```

end enrich

```

string-data =
data specification
  using char
  string = ""
    | . + . prio 9 (. .char1 : char ; . .rstring : string ;) prio 9
    ;
  variables str, str0, str1, str2, str3: string;
end data specification

```

```

char =
specification
  sorts char;
  variables char, char0, char1, char2: char;

```

axioms

```

char generated by "a", "b", "c", "d", "e", "f", "g", "h", "i", "j",
"k", "l", "m", "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x",
"y", "z", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K",
"L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W",
"X", "Y", "Z", "!", "@", "#", "$", "%", "^", "&", "*", "_", "-",
"+", "=", "~", "<", ">", "?", ".", ":", "/", "|", "0", "1", "2",
"3", "4", "5", "6", "7", "8", "9", "(", ")", "[", "];

```

end specification

```

nat-basic2 =
enrich nat-basic1 with
  functions . + . : nat × nat → nat prio 9;
  variables m, n0: nat;

```

axioms

```

  n + 0 = n;
  m + n + 1 = (m + n) + 1;
  n < n0 ∨ n = n0 ∨ n0 < n;
  1 = 0 + 1;
  0 ≠ 1;

```

end enrich

```

nat-basic1 =
data specification
  nat = 0
    | . + 1 (. - 1 : nat ;)
    ;
  variables n: nat;
  order predicates . < . : nat × nat;
end data specification

```
